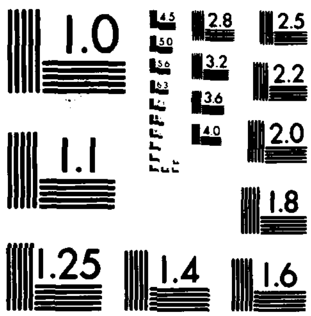
AD-A091 029 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
ON MUTATION.(U)
AUG 80 A T ACREE DAAG29-80-C-0120
UNCLASSIFIED GIT-ICS-80/12 NL

1 of 2



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Information

(12) LEVEL II

(14) GIT-ICS-80/12

(6) ON MUTATION+

(10) Allen Troy Acree, Jr*

(9) Doctoral thesis,

(12) 184

(11) August 1980

*School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

+Work supported in part by U.S. Army Research Office, Grant #DAAG29-80-C-0120,
and by Office of Naval Research, Grant #N00014-79-C-0231.

(15) N00014-79

411044

JOB

ON MUTATION

A THESIS

Presented to

The Faculty of the Division of Graduate Studies

by

Allen Troy Acree, Jr.

In Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

in the School of Information and Computer Science

Georgia Institute of Technology

August, 1980

ON MUTATION

Approved:

Richard DeMillo, Chairman

Albert Badre

Lucio Chiaraviglio

Richard LeBlanc

Frederick Sayward

Date approved by Chairman 8/29/80

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The author would like to thank the Graduate Division for the waiver of certain format requirements so that this thesis could be prepared on the PRIME 400 under the RUNOFF text processing system. The author also wishes to express his gratitude to Prof. Frederick Sayward for his helpful comments on the preliminary design of the Cobol Mutation System, as well as for advice on the management of its implementation, and to Jeanne Hanks for her help in implementing the system and its later functional enhancements, and especially to his advisor, Prof. Richard DeMillo, for his thoughtful guidance and encouragement. This research was supported in part by the US Army Institute for Research in Management Information and Computer Science, ARO Grant No. DAAG22-78-0121 and the Office of Naval Research, Grant No. N00014-79-C-0231.

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER I

INTRODUCTION

Program testing has been practiced as long as has programming itself, in spite of the general confession that testing can never prove in any absolute sense that a program is correct. Two facts are responsible for the popularity of testing. The first is that testing has a tendency to uncover program errors, and that the more systematic the testing, the stronger this tendency. The second is that a program that is not completely correct is not necessarily unreliable in a given operating environment, and that even a program that is not completely reliable will usually not be completely worthless to its users. Those responsible for software system development are charged with deciding how much they are willing to pay for a given increase in reliability. The challenge for research is therefore to produce a testing method that is (1) more effective at uncovering errors and (2) less expensive to apply. Mutation analysis has been put forward as such a method [1,11,12,5]. Working mutation systems have demonstrated that mutation analysis can be performed at an attractive cost on realistic programs. (See Appendices A-D.) In this work, the effectiveness of the method is studied by experiments with

programs in the target application spaces. Most of our target programs are in Cobol. Cobol was chosen as a language of study for several reasons. A pilot system had already been implemented for Fortran [5,11], and preliminary results on testing small numerical subroutines were encouraging. A more complete Fortran system was being developed concurrently with the development of the Cobol system on which this work is based. We were interested in knowing if the mutation concept would be as useful in a language like Cobol as it had been in Fortran, with Cobol's different concepts of data structures, and with input and output, which had never been included in the Fortran systems. [19] For a description of the Cobol system and its treatment of the data division and input and output, see Appendix A. We were also interested in a system that would allow us to collect empirical data on programming and testing practice and effectiveness. Since Cobol is widely used, many programs are available for study. Since programming in Cobol is often done under strict regimentation, it was expected that we can obtain complete packages consisting of programs along with their test data and error histories.

Software system development has been described in [22] as a sequence of steps leading from problem definition to software, with corresponding validation tasks relating the result of each step to previous steps. The major steps are

(1) System requirements definition

(2) System functional specifications

(3) Software requirements definition

(4) Software functional specifications

(5) Software implementation

The mutation analysis methodology examined in this work has as its goal validation of the last stage, software implementation. As such it overlaps some proposed validation methods, and complements others. The following sections outline some of these techniques.

Automated Aids for Software Validation

The present work deals with mutation analysis, which is an automated aid for software validation. It is useful to survey several such aids designed for related purposes. All of these tools have as their goal an increase in confidence that a given software product will function as desired under normal operating conditions.

Static Code Examination Tools.

The software can be examined statically (without execution) for some types of errors.

Syntax Checkers - Compilers. The use of a compiler to detect syntax errors is so common that we usually do not think of it as a validation tool. The errors that are detectable by a simple syntax check are usually limited to

those such as the use of a variable of one type where another type is required, or the misspelling of a variable name, resulting in an undeclared variable, or parameter mismatch in subroutine calls [34]. Languages such as Fortran that permit implicitly declared variables and separate subroutine compilation restrict the amount of error-detection that a compiler can do.

Standards Enforcers. Some Fortran compilers can be invoked with optional parameters that force the compiler to treat undeclared variables as errors [28]. This is an example of the use of automatic verification of extra-syntactic rules called standards that are thought to be useful in avoiding the introduction of errors into software in the first place. These standards may have the form of additional syntax rules (e.g. all variables must be declared), the deletion of otherwise legal program constructs (e.g. ALTER or GOTO), or naming or documentation conventions.

Structural Analysis. A more sophisticated form af static analysis can give some information about the dynamic behavior of a piece of software. Structural analysis by a system such as DAVE [24,25] can produce diagnoses such as

(1) The variable X is referenced before it is defined along all flows of control in the module. (Always indicates an error.)

(2) The variable X is referenced before it is defined along some flows of control in the module. (Indicates an error, if any of those control paths are actually executable.)

(3) The variable X is defined but not later referenced along any control path. (This indicates an inefficiency, at best, and more likely a design flaw.)

The Path Analysis strategy studied by Howden [16] is an attempt to partition test cases into domains, each of which forces the execution of some particular logical path through the program.

Dynamic Evaluation Tools

In principle, anything that can be learned about a program can be inferred from the code and the environment in which it is to be run. However, it is usually more economical to stop looking at the program at some point and start looking at its results. We can imagine programs whose input domains are small finite sets. Such programs can be completely validated by exaustive testing. However, in practice this class of programs is so small that exaustive testing is usually not a useful option.

Random or Partially Random Test Data. Tests with

randomly generated test data are appealing because of their ease of implementation. One would like, for instance, to be able to specify a probability distribution on the inputs of a program and automatically generate a test data set of the desired size. If the distribution of inputs in the software system's actual operation environment is known, one could then actually estimate the statistical reliability of the software. Here "reliability" means the probability that the software will function in its operating environment for a given period of time without failure [11,13]. However, in practice the distribution of inputs is often not known, so random testing does not then produce a reliability estimate. The main problem with random testing is that just doing more of it may not necessarily increase confidence in the program by much. A hundred random test cases may test a few sections of the program a hundred times, rather than testing a hundred sections of the program. The following small experiment illustrates this point.

The experiment was performed to measure the effects of program choice, test data selection method, and data set size on the adequacy of test coverage. The coverage measure used was the mutation score from the first Fortran mutation system. The mutation score will be discussed fully in Chapter II, but for now it is sufficient to know that the scores range from 0.0 to 100.0, with the higher score indicating more complete test coverage. Two programs called

JBST03 and JBST05, first reported in [7], were used in the experiment. They are both sorting programs so that the same test data may be used, but they are based on different algorithms. The test data selection methods are random (from a table of random digits), and hand selection. All of the hand-selected data was chosen before any testing was performed, on the basis of a general knowledge of sorting. The small test sets were composed of three vectors, of lengths 1, 5, and 10. The large test sets contained six arrays, of lengths 1, 2, 5, 6, 8, and 10. Two replicates of each combination were generated, and the mutation scores were measured. The results appear in Table 1.

Table 1. Effects of Test Size, Selection Method, and Program on Test Adequacy

	PROGRAM Mutation Scores		Test Set
	JBST03	JBST05	
Hand Selection	95.6	92.5	small
	96.3	92.2	
	96.7	95.2	large
	96.7	95.2	
Random Selection	96.3	94.9	small
	96.3	93.8	
	96.7	94.7	large
	96.7	94.8	

The effects are small, since all of the test cases score in the 90-100 range, but there are strong

statistically identifiable effects. Table 2 is an analysis of variance table, with effects a=program, b=data generation method, and c=test case size. (See Appendix E for a short discussion of analysis of variance.)

Table 2. ANOVA Table for Size-Selection-Program Experiment

Effect	Estimate	SS	df	MS	F
a	-2.23	19.80	1	19.80	176 *
b	-0.50	1.01	1	1.01	8.98
ab	-0.32	0.41	1	0.41	3.64
c	+1.10	4.84	1	4.84	43.02 *
ac	+0.55	1.20	1	1.20	10.67
bc	+0.71	2.02	1	2.02	17.96 *
abc	+0.53	1.11	1	1.11	9.87
SSE		0.90	8	0.1125	
SST		31.29	15		

The effects marked with an asterisk are significant at the 0.005 level. Thus we see that the program being tested is a major source of variation. Despite the fact that the programs perform the same function, one is more easliy tested than the other. The size of the test set is also important, with larger test cases providing better coverage on a given program. Neither of these conclusions is surprising. Since the b effect is not highly significant, hand selection and random selection did not produce very different results in this range of sampling. However, the significance of the bc interaction leads us to believe that as the size of the test data set increases, hand-selected test data improves its performance faster than does randomly

selected test data. Thus random test data may be less desirable than test data that has been selected according to some plan that takes into account properties of programs.

Symbolic execution. One measure of test data effectiveness is the number of different control paths that the test data will cause to be executed. The ATTEST system described in [8,9] analyzes the structure of a program and develops symbolic requirements for the traversal of given paths in the program. As the name "symbolic execution" suggests, the system steps through the program accumulating symbolic expressions rather than the usual numerical values. A branch condition results in a conditional expression involving algebraic formulas. At the end of a path, then, a compound logical expression involving algebraic expressions in the input variables is obtained. These expressions may then be solved automatically for input values that will drive execution down the desired path. Symbolic execution systems for subsets of LISP [4] and PL1 [18] have been reported.

Program Instrumentation. As was mentioned in the preceeding paragraph, coverage of program paths is a measure of test effectiveness. This measure can be used as a driving criterion for test data selection, or it can be evaluated for test data generated by arbitrary processes. The evaluation of the coverage measure can be implemented by instrumenting the program; that is, inserting instructions

that do not affect the functional behavior, but which use auxilliary variables to keep track of program behavior. Instrumentation can be used for paths of arbitrary complexity, but is most often limited to simple Decision to Decision Paths (DDP's) [27,15] and "hidden paths" [11] within predicates. While many more sophisticated techniques are being studied, the DDP method is widely available at the commercial level [2,30]. However, examples of simple errors that could escape detection by the DDP procedure have been reported in [14].

Mutation Analysis. Mutation Analysis produces a measure of test data effectiveness that includes simple DDP coverage but is much more comprehensive. Test data that receives a high mutation analysis score must not only force the execution of all program statements, but must also demonstrate to a high degree of confidence the correctness of the operations along the paths. However mutation analysis systems do not automatically generate test data. But the listing of live mutants is generally very helpful to the human tester in devising test cases. A full discussion is deferred until later sections.

Other Approaches to Software Validation

Formal Verification. Formal verification has been proposed in [20] as the ultimate program validation technique. In this technique the tester is required to produce a mathematical proof that the program's behavior is

consistent with its functional requirements. Manual theorem proving for programs is usally such a large process that the technique depends on the availability of automatic theorem provers, or at least semiautomatic ones that pause occasionally and ask for advice. Another requirement is that the statements in the programming language have their semantics expressible in simple axioms. The key reservations that many researchers have about formal verification are

(1) Can formal verification be made practical for large software systems? This depends on developing very efficient (in both space and time) theorem provers. W.D. Maurer recently reported in [21] that verification of a two page Cobol program was obtained at the cost of $10,000. Mr. Maurer was speaking in favor of verification.

(2) Can formal verification be made sufficiently reliable? At the present "proofs" of programs are as subject to error just as are the programs themselves [10,14]. Reliability may be improved by improving the reliability of automatic tools.

(3) Can production software be formally specified as completely as is required for formal

verification. Testing does not usually require a complete prior specification.

Error Seeding. Error seeding [13] treats the program as a statistical object. A known number of errors are deliberately introduced into the program, and testing proceeds until a predetermined number of errors have been discovered. If all errors are random and independent, one could use the ratio of seeded to nonseeded errors among those discovered to estimate the total number of errors remaining. This is a direct analogy to common wildlife population estimation techniques. The problem is that experience shows that errors are not random objects [1], and their clustering and dependent behavior may spoil this analysis.

CHAPTER II

CONCEPTS OF MUTATION ANALYSIS

Conditional Correctness

The chief concept underlying mutation analysis is that of conditional correctness.

Given:

a program P,

a class of programs M; $P \in M$,

evidence E about the program P.

Conclude:

If a correct program P' is in M then either P is correct or E demonstrates the incorrectness of P.

This paradigm is satisfied, for example, in the case of M being the set of programs for evaluating polynomials of degree < 5. Then E is the evaluation of P on 5 distinct points. Given that the desired program is in fact in M, E is sufficient to decide whether or not P is correct. In this example, E is sufficient to distinguish any two elements of M. In the more general case, this need not

hold. All that is necessary is that E distinguish P from every element of M that is not equivalent to P. We say that two programs are <u>equivalent</u> if they have the same input-output behavior. We say that an element of M1 or M2 is equivalent (or nonequivalent) if it is equivalent (or, respectively, not equivalent) to P. This result has been extended to much wider classes of programs, but those extensions are still based on polynomial behavior [29].

Now consider a slightly more complicated situation.

Given:

a program P,

two classes of programs M1 and M2; with $P \in M1 \subsetneq M2$,

evidence E about the program P.

Conclude:

If

(a) there is a correct program P' in M2 and

(b) whenever E distinguishes P from all of the nonequivalent programs in M1, that E also distinguishes P from all of the nonequivalent programs in M2;

then either P is correct or E demonstrates the incorrectness of P.

It is noted that the second situation is

mathematically isomorphic to the first (M1 is redundant.) However, we will be interested in the experimental situation in which property (b) does not actually hold completely, but is rather a statistical description.

Mutagenic Operators

Mutation analysis is an implementation of conditional correctness where P is a program written in some programming language and M1 is a set of mutants of P. A mutant of P is a program derived from P by making a single, simple source language change in the program. Mutations are produced by mutagenic operators such as:

> (in Cobol) Reverse any two adjacent elementary items in a record.
>
> (in Fortran) Reverse the dimensional limits in a two-dimensional array.
>
> (in any language) Substitute for a reference to a variable a reference to any other variable appearing in the program.

The choice of mutagenic operators is influenced by three concerns:

> (1) to include most common programming errors [11,32].
>
> (2) to obtain program coverage by including

special operators that indicate whether or not statements have been executed, and whether or not those executions had any effect on the final result.

(3) to permit straightforward and efficient implementation in an interpretive or compiled system.

Evidence E results from executing P and some of its mutants on a set of test data. The strength of the evidence is to some degree under the control of the designer of the mutation system. If the set of mutagenic operators implemented in a system allows test data to pass mutation analysis (distinguish P from all of M1), and important errors are not detected, then the set of operators can be augmented, adding programs to M1 and strengthing the evidence, by forcing the user to provide stronger test data. Similarly, if operators are found to be of little use in testing (adding little strength to the test evidence), then those operators may be deleted. Operator selection be discussed further under the proposed experiments.

The Competent Programmer Assumption and The Coupling Effect

For any realistic choice of M2, either assumption (a) or (b), or both, will not be fully satisfied.

For example, let M2 be the set of programs which a

programmer might produce in the course of an effort to produce a program P which satisfies functional requirements f. Then, just assuming that the programmer could possibly write a correct program, assumption (a) will be satisfied. But assumption (b) is probably not. For any program P and any finite test set E it is possible to find some other program P' such that P and P' agree on E but nowhere else. If both P and P' are possible results of the programming practice, then (b) will fail.

At the other extreme, let M2 = M1. Then assumption (b) is trivially satisfied, but (a) is not, since we know by experience (Appendix D) that even the best programmers produce programs that contain errors more pervasive than a single, simple change. Another way to view this is that it often takes more than a single change to correct a "buggy" program. (See for example a discussion of a program by Naur in [14].)

In mutation analysis, we try to balance the two assumptions and choose an M2 so that neither is dramatically false. Even so, the definition of M2 is rather vague. Generally we choose M2 to be the set of programs that are "close" to P in a syntactic sense. M2 would contain multiple mutations, as well as perhaps simple missing path errors, etc. Assumption (a) is called the <u>competent programmer assumption</u> [11,1]:

A competent programmer, after completing the

> iterative process and deeming that his job of designing, coding, and testing is complete, has written a program that is either correct or is almost correct in that it differs from a correct program in "simple" ways.

Assumption (b) is called the coupling hypothesis [11]:

> Test data that is sensitive enough to detect all simple errors is sensitive enough to detect most likely complex errors as well.

If the competent programmer assumption and the coupling hypothesis were completely valid, then mutation analysis would be a perfect testing technique. Since elimination of all simple errors would eliminate all possible errors. This work addresses the coupling hypothesis, and attempts to place statistical bounds on its validity.

The following is one possible definition of a general "coupling effect".

> Let P be a program, M1 a set of programs, and M2 another set of programs. We say that M2 is coupled to M1 (for P) if whenever a set of test data T distinguishes P from all of the nonequivalent members of M1, then T also distinguishes P from all of the nonequivalent members of M2.

The existance of a coupling effect of this type has been proved in [6] for decision table programs where M1 = {single mutations of P} and M2 = {multiple mutations of P}. In the more usual setting of Fortran and Cobol programs with M1 = {single mutations} and M2 = {all likely errors}, then the strong form of the coupling effect does not exist, since multiple mutations can escape detection by test data that are sufficient to detect first order mutations. This problem will be addressed specifically in Chapter III. These uncoupled errors, or likely programming errors that are not detected by test data generated for first order mutation analysis, will be collected from the experiments, and studied to see if they suggest new mutagenic operators to be added to our current set in order to strengthen mutation analysis.

We can however express the coupling effect empirically:

> Let P be a program, M1 a set of programs, and M2 another set of programs. We say that M2 is coupled to M1 (for P) with coupling coefficient (1-w) if w is the largest number such that:
>
> for any T distinguishing P from all nonequivalent elements of M1, the number of elements of M2 that

are nonequivalent and not distinguished by T is not greater than w|M2|.

Examining all possible test cases is not in general possible (else there would be no need for any other testing methodologies), so this definition is operationally deficient. We can however define another coefficient z to be the fraction of the nonequivalent members of M2 not eliminated by some particular test case. z is then a random variable over the space of program/M1-sufficient test-case pairs, whose upper bound is w. An experiment on the coupling effect is a measurement of the strength of that effect by measuring z, and hence estimating w. Actually, z itself would only be estimated by sampling. A confidence interval (see Appendix E) could be determined for z. The conclusion of such an experiment could be of the form:

> For programs selected from population Q and test data generated by process R (to a strength sufficient for first order mutation analysis) the values of z were estimated by sampling from the sets M2 generated by process S and were found to range from x to y.

Thus if Q is similar to a population of programs about which we want to make quantitative testing statements, and R is the testing procedure that we want to quantify, and S

generates a reasonable distribution of cantidate alternative programs, we can use the estimated values of z to bound the likelihood that errors remain in a program.

The validity of the mutation analysis technique thus rests on the competent programmer assumption and the coupling effect. The major effort in this research is toward finding the strength of the coupling effect, and thus toward finding a limit on the reliability af mutation analysis.

Equivalence of Mutants

Not all first order mutants can be eliminated, no matter what test data is supplied, since some mutant programs will be functionally identical to the original program. Some of these equivalent mutants can be detected automatically, with methods borrowed from code optimization theory [3,1]. For example, changing

```
A := 0              A := 0
           ==>
B := 0              B := A
```

is an equivalent mutation that can be detected at compile time and eliminated (i.e. not generated). Since equivalence is formally undecidable, we can never hope to detect all of them this way. Mutation systems will continue to rely on the human user to judge the equivalence of some mutants. The accuracy of the typical user in judging equivalence needs measurement, as does the cost of improperly judging a mutant equivalent when in fact it

represents a potential error.

Most equivalent mutants encountered in testing are very simple ones, like the example above. Another major source of simple equivalent mutants is the inclusion in a program of useless variable initializations. If a program includes "A:=0", and each possible execution path has another assignment to A before A is used, then the "0" in "A:=0" may be changed to anything else. Or the A may be changed to any other variable that does not need a nonzero value at that point. An example of a useless initialization in a Cobol program used in this study is

```
MOVE SPACES TO PRINT-LINE.
WRITE PRINT-LINE FROM HEADER-LINE AFTER PAGE.
```

Another source of equivalence is assignments that "almost" don't matter. For example, if in a Cobol program FLAG is used as a boolean with 'TRUE' for true and 'FALSE' for false, and the only test in the program is IF FLAG = 'TRUE'... then an assignment FLAG = 'FALSE' can be changed to FLAG = 'HELLO', or anything else other than 'TRUE'. A statement such as MOVE ZERO TO NUM-1, where NUM-1 is defined to have no fractional part (e.g. PIC 99.), can be changed to MOVE 0.12 TO NUM-1, due to the Cobol rules for numeric truncation in a MOVE. The detection of equivalence in other cases may not be so easy. Changing IF A = 11 to IF A IS NOT < 11 may not be judged equivalent until analysis of the program shows that A can never be greater than eleven at

that point. Obviously, examples of arbitrary complexity may be constructed.

CHAPTER III

THE COBOL MUTATION SYSTEM

Design and History of Mutation Systems

Automated systems to aid mutation analysis have been developed [1,5,11,12,19]. Such systems are composed of the following basic functions.

(1) A parser to reduce the source code to an internal form suitable for interpretive execution and mutation.

(2) A mutation generator that produces a list of mutation descriptions applicable to the program, based on its internal form.

(3) An interpreter that executes the program or a mutant program on a test case and records the results of execution.

(4) A test data handler and user interface to provide a convenient software test harness. This allows the user to submit test cases, examine the results, and either reject the test case or accept

it for further analysis.

(5) A mutator that modifies the internal form in such a way as to correspond to a source language error, and later restores the program to its internal form.

(6) A report generator that summarizes information to the users terminal and to a permanent file in which is stored the status of the mutation analysis, the mutants remaining, and the test cases.

The first automated mutation system was FMS.1 (for Fortran Mutation System -- version 1) developed at Yale University [11]. FMS.1 was developed on a PDP 10 and was later transported to a PRIME 400 at Georgia Tech, a DEC 20 at Yale University, and a VAX 11 at the University of California, Berkeley. FMS.1 treats only a subset of Fortran: a single subroutine with integer arithmetic and without I/O. Success with this pilot system was sufficient to motivate the construction of more elaborate systems.

FMS.2 was also developed at Yale and transported to Georgia Tech. It accepts multiple subprograms in full ANSI Fortran (minus I/O) [19,1]. FMS.1 is less of a user-oriented system than FMS.1, and was designed primarily

to allow the flexible design of mutation experiments.

CMS.1, a mutation system for Cobol, was designed at Georgia Tech by the author and implemented on the Georgia Tech PRIME 400. The design owes much to the earlier FMS.1, as well as to discussions with its designers. For a full discussion of this system, see appendices A,B, and C.

A Case Study

During the development of CMS.1 the author had difficulty debugging a subroutine called NXTLIV. Since CMS.1 is written in Fortran, it was decided to test the subroutine under FMS.1. (FMS.2 was not then available at Georgia Tech.) It was necessary to modify the subroutine somewhat in order to conform to the FMS.1 Fortran subset, but it was felt that the error(s) probably did not lie in the code that required modification. A condensed script of the testing session appears as Appendix D. One error was found quickly. The ease of finding the error is probably due less to mutation itself than to the convenient subroutine test harness provided by FMS.1. A second error was found later, however, as a direct result of trying to eliminate one of the last remaining mutants. An interesting note is that the mutant being considered was not the correction of the error, but another mutant yet to be considered was. This is an example of the coupling effect. Detection of one potential error automatically detected another.

Programs Used in This Study

Most of the experiments reported here use data generated from six Cobol programs obtained from several sources. Each of the programs was modified slightly to fit in the CMS.1 Cobol subset. One typical modification was the replacement of a serial disjunction of the form

IF A = 'A' OR 'C' OR 'Q'

by the equivalent form

IF A = 'A' OR A = 'C' OR A = 'Q'

Another is the replacement of a condition name by its defining condition. In some programs record sizes were reduced without affecting program logic. Listings of the programs as tested may be found in Appendix F.

Program 1 is from the Army SIDPERS personnel system, and contains 146 lines of code. In its original form there were otptional sections for different input forms (disk and tape) and different output dispositions (disk and printer). These options were deleted to conform to the CMS.1 sequential input - sequential output restriction. The deleted code is essentially a copy of retained code with different options on the READ and WRITE statements. No errors were found in this program during the experiments. The progam has two input files, both containing a key and information field. The files are presumed sorted on the key fields, and represent old and new master files. The program produces a log of the differences between its two input

files. Program 1 is used to illustrate the use of CMS.1 in Appendix C.

Program 2 contains 163 lines of code and was written by a student at Georgia Tech as an exercise. The program accepts account transactions and performs one of several simple computations based on a class code in the input record. Data validation is performed, and the output consists of one record for each input transaction, plus summary statistics by class.

Program 3 is adapted from Learning to Program in Structured Cobol [33]. Input transactions are in the form of pairs of records. For each pair the first record is a name-address-phone-account-number record, and the second contains credit information. From that credit information discretionary income is computed by a standard formula. The purpose of the program a readable listing of the input file with name and address in one column and decoded credit information in another. One small error was found; there was code to handle the situation of an end-of-file after the first card of a pair, but this code did not bring execution to a graceful end. Instead, the program terminated abnormally several statements later when another READ was attempted. There were also several useless initializations. Such useless statements are a nuisance in mutation analysis since they can be changed to any other useless statement without affecting the input-output behavior of the program.

Program 4 is adapted from ANS Cobol: A Pragmatic Approach [26] where it is called SRMFREP. The input records are codings of student academic data, including name, address, major, status, and a number of course items consisting of the department, credit, and grade for each course taken. The program computes the students' grade point averages and produces a listing with name, address, and other information in one column, and three columns of course reports. The original program was written to accept very long input records (>1000 characters). Since CMS.1 allows a maximum of 150 characters per record, some abbreviation was necessary. The identifying fields were shortened, and the maximum number of course reports reduced to 11. One error was found; code to handle invalid input records could sometimes refer to undefined data fields.

Program 5 was also written by a student at Georgia Tech. Input transactions contain identifying codes for a store, a department, and a salesman. The salesman's name, year-to-date sales, current sales, commission rate, and months employed are also included. In the computation, commision bonuses are paid, depending on the department and the average sales volume. Some data validation is performed, and error report records are interspersed with valid transaction report records. One functional error was discovered during testing. If a page-full condition is raised by the printing of an error report, then no heading

would be generated for the following page. Several data flow anomolies, such as useless initializations, were detected.

Program 6 is also taken from Learning to Program in Structured Cobol [33], and was written as an extension to Program 3. In addition to computing discretionary income, a credit limit is computed based on discretionary income, marital status, home ownership, and job tenure. Rather than just creating a listing from its input, the program uses the input as transactions against a master file. The input and master files are presumed to be sorted by account number, and a new master is produced. A separate log of transactions and errors is also generated. The transaction types are add, delete, and change master records. This program was apparently not tested before publication, since it did not function properly on any input. Faulty program logic caused the last transaction card-pair to be ignored. An empty transaction file caused abnormal termination. The input is validated in one section of the program, but not in another similar section. If the first card pair is an invalid transaction, the error message is placed in the log file before the log file header. Many extra initializations and data field definitions are present, due largely to the free use of the COPY verb. The program, after correction, contains 619 lines.

Test Data Generation

Test data for use in the experiments was generated in the way in which we would expect such data to be generated in production use of a mutation system. A tester (in this case the author) first manually generated tests to cover the major points of the specification. For example, if a program is supposed to produce one type of record for a zero input field and another type if the field is nonzero, the test data would include both. Actually this initial test data does not even have to be very good, because of the feedback supplied by the mutation system. The tester enbles a subset of the mutants, and starts a mutation run. The mutants alive (i.e. not eliminated

not differentiated fron the original program) at the end of the run suggest new test data that the tester must generate. This cycle continues until all nonequivalent mutants have been eliminated. Then a larger subset of mutants is enabled. Testing continues as before until all nonequivalent mutants are eliminated. The subsets used in this study are

1) The TRAP mutants. Elimination of these requires that all statements in the program be executed.

2) A random 10% of all substitution mutants, and all of the other types. This seems to yield strong test data with reduced computational effort [1].

3) All mutants that can be generated by the system.

(See Appendix A for a list of the mutagenic operators

supported by CMS.1.)

Program Statistics

The results of mutation analysis on the six programs is summarize in Table 3, which shows for each of the six programs the number of program lines, the number of mutants when the substitution mutants are generated with probability 0.1, the number of those mutants equivalent to the original program, the total number of mutants that can be generated, and the number of those that are equivalent.

Table 3. Mutation Statistics on the Six Programs

Program	number lines	number mutants at 10% *	number equiv. at 10%	number mutants at 100%	number equiv. at 100%
1	146	389	17	1098	21
2	163	603	36	2814	47
3	238	1125	61	6340	106
4	321	1609	58	7334	95
5	455	1527	92	7957	228
6	619	4011	128	28275	428

* 10% of substitution mutants, 100% of other types.

Empirical Complexity of Mutation Analysis

With the operators now in use in the various mutation systems, it has been seen that the number of mutants of a given program is approximately proportional to the square of the length of the program [1]. For Cobol programs perhaps a better estimator of the number of mutants is the product of

the data division length and the procedure division length. Indeed we can almost predict such an empirical law from first principles. Some of the mutant types are inherently bounded by linear growth in the program size. Examples would be arithmetic operator substitutions, in which there are a fixed number of substitutions to be made for each occurrence of an operator in the program. The number of such source operations is no more than the length in characters of the source program. The dominant mutant types, for large programs, are the operand substitution types [1]. The number of those is bounded by the number of data references in the program times the number of distinct data items to be referenced. Both of those are bounded by the length of the program (or for Cobol, by the length of the procedure division and the data division, respectively.) Figure 1 plots the logarithm of the program size in lines against the logarithm of the number of mutants from Table 3. Since the points seem to lie about a straight line with slope 1/2, we see that the number of mutants is quadratic in program size. The graph also shows the number of equivalent mutants for the programs. We see that the number of equivalent mutants is also quadratic in program size. This could be troublesome for larger programs unless most equivalent mutants can be detected automatically.

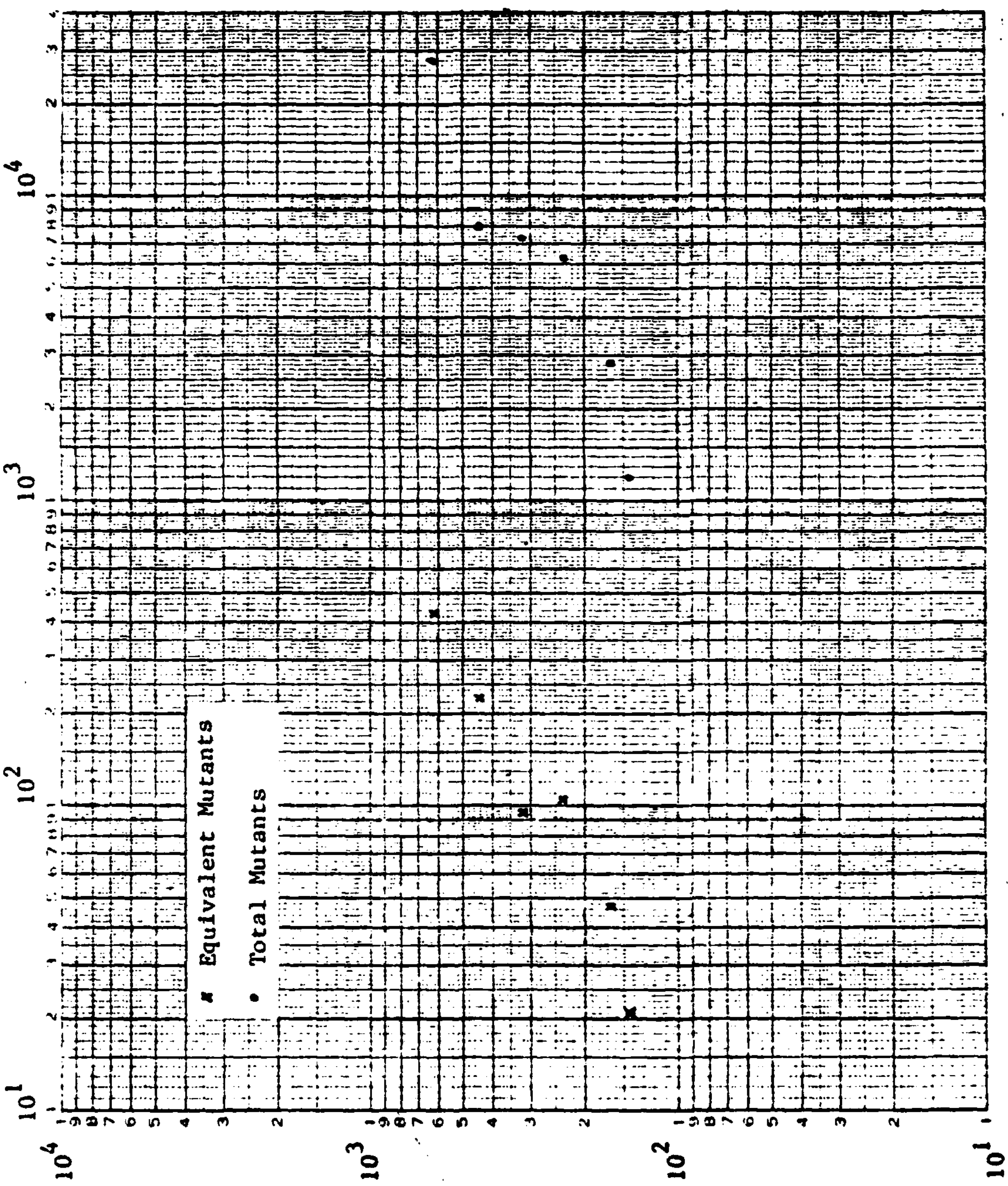


Figure 1. Log-Log Graph of Program Size vs. Number of Mutants and Number Equivalent

CHAPTER IV

EXPERIMENTS ON THE COUPLING HYPOTHESIS

Empirical evidence has been found [1] for the coupling effect for Fortran programs, but this evidence is weak in that only a very few programs have been studied in a limited way. This research will extend these results by more extensive studies in an attempt to place bounds on the statistical validity of the coupling effect.

A series of experiments has been devised to test the hypothesis that testing a program to a degree sufficient to eliminate first order mutations is necessarily also sufficient to eliminate most likely complex mutations as well. The experiments all have the same basic format:

Step 1: for a given program, generate test data using a mutation analysis system, sufficient for first order mutation.

Step 2: Randomly generate a large number of more complex mutants, execute the resulting programs on the test data from step 1, and list mutants not eliminated.

Step 3: Manually examine the list to remove equivalent mutants.

In step 2 in all cases, we use uniform sampling with replacement from a given space of complex mutants. Thus the

parameters of each experiment are the program being tested, the tester, the type of complex mutants considered, and the sample size. These experiments were performed using a single tester (the author), and a single set of test data for each program. The repetition of these experiments by other investigators would enable us to estimate the variation in the coupling effect due to test data generation.

Random Pairs of First Order Mutants

One place to start looking at the coupling effect is with "complex errors" defined as pairs of simple mutants. It is not reasonable to look at all possible pairs of mutants because of their number. A small sample program might have on the order of ten thousand mutants, giving a hundred million mutant pairs. (Actually the number would be somewhat less, since not all pairs are possible, but the order of magnitude is correct.) It is quite feasible to run that many mutants, but the number of mutants that must then be examined by hand for equivalence is unmanagable. We can obtain sufficient information by selecting a reasonable number (in this case 50,000) mutant pairs from one program, and then selecting more from a different program, and so forth. Sampling programs as well as mutants will make any conclusions more general. When the coupling effect is total (w=1.0), test data developed to eliminate all first order nonequivalent mutants eliminates all higher-order

nonequivalent mutants as well. Since the coupling effect is not expected to be total in practice, what we need is a confidence interval on the fraction of second order mutants that are not equivalent and are not eliminated by data chosen to eliminate first order mutants. If we find any such "bad" second order mutants, we can obtain a two-sided confidence interval on that fraction (see Appendix E). If we find none, then we can still obtain a one-sided (upper bound) confidence interval. This will give us an estimate of the probability that an error of the type {second order mutation} would escape detection in mutation analysis. For this experiment pairs of mutants were selected uniformly from the list of first order mutants, by a pseudo-random number gererator. There were some technical difficulties. A mutant is a mutant of a <u>particular</u> program, and may not have meaning for another. In particular, if S and T are mutations to a program P, producing programs S(P) and T(P), then T(S(P)) may not necessarily be a legitimate mutant of S(P). For example, if S is "Delete statement 27" and T is "In statement 27 replace I by J", then T cannot follow S. So in the selection procedure such things had to be avoided. The method was to select a pair of mutations, check their validity as a pair, and make the mutation if valid. Invalid pairs were discarded. The process continued until the required number of valid pairs had been selected. The results are summarized in Table 4.

Table 4. 50,000 Random Pairs of Mutants for Each Program

Program	Pairs Survive 1st Order Test Data	Not Equiv.	95% Confidence Interval on (z * 100,000)**
1	26	0	0.0 -- 7.4
2	12	0	0.0 -- 7.4
3	22	5	3.2 -- 23.3
4	10	2	0.5 -- 14.4
5	45	0	0.0 -- 7.4
6	13	0	0.0 -- 7.4

** z is the probability that a randomly selected pair of simple mutants would generate an uncoupled complex error for this test data.

The numbers are very favorable for mutation analysis. Test data generated to be sufficient for first order mutants proved to be sufficient for at least 99.976% of all second order mutants in all cases considered, and 99.992% in most cases. These results can be stated in several ways. In the terminology of Chapter II, the coefficient of coupling of the set {first order mutants} to the set {second order mutants} for a given program is very close to unity. Significantly, program size does not seem to be an important factor in the coefficient. In terms of implications for the design of mutation analysis systems, the addition of second order mutations gives almost no power not already present in first order mutations, and certainly not enough to justify their cost.

However, uncoupled mutants were found in the experiment, and they may lead to insights into how mutation analysis may be strengthened in other dimensions, such as the choice of first order mutagenic operators. All of the uncoupled mutants found were pairs of alterations to a predicate; either changing a comparison operator and one of its operands (Type A), or changing both operands of a comparison operator (Type B). There were four type A mutants, one of which is

```
IF(MARITAL-STATUS-WS = 'S')
          ==>
    IF(NAME-L1 < 'S')
```

and three type B mutants, like

```
IF(SOC-SEC-IN NOT = '999999999')
              ==>
 IF(ADDR-IN-2 NOT = SOC-SEC-F1)
```

If we treat the uncoupled mutation as a potential error (or correction) to the program, then they represent a form of coincidental correctness: taking the right path for the wrong reason.

Correlated Pairs of First Order Mutants

It has been suggested [1] that completely random and independent sampling is not really a fair test of the coupling effect. Most single mutants are unstable and are eliminated rather easily, and so random pairs will be even more unstable. Perhaps we should look not at independent pairs, but rather at pairs of errors that have a chance of producing subtle errors. Those would be pairs of mutations

that "almost cancel". We can develop the capability of automatically generating "correlated" mutant pairs. A proposed criterion for such pairs is that they either refer to the same variable or to the same statement. A weaker restriction would be that they refer to statements that reference the same variable. Note that all of the uncoupled errors from the previous experiment fit this criterion. The procedure for pair selection is to randomly select a pair of substitution mutants, and check to see if they reference statements which reference the same data item (either a variable or a constant). Pairs that alter the same reference in the same statement are not considered, since they are in effect first order mutations. The procedure is repeated until 10,000 correlated pairs are generated and tested for each program. The results are presented in Table 5, where for each program, 10,000 correlated mutant pairs were created.

Table 5. 10,000 Correlated Pairs of Mutants for Each Program

Program	Pairs Survive 1st Order Test Data	Not Equiv.	95% Confidence Interval on (z * 100,000)**
1	0	0	0.0 -- 36.9
2	3	1	0.3 -- 55.7
3	60	19	114.4 -- 296.6
4	3	3	6.1 -- 87.6
5	1	0	0.0 -- 36.9
6	1	0	0.0 -- 36.9

** z is again the probability that a randomly selected complex mutant of the current type would represent an uncoupled error for the given test data.

Eighteen of the uncoupled mutants are of Type A, defined in the previous section. Four are of Type B. The other uncoupled mutant is also a pair of mutations to a conditional expression, but the two mutations do not affect the same comparison. The complex mutation is

IF(ACCOUNT-NUM IS NUMERIC AND BILLED-AMOUNT IS NUMERIC AND...

is changed to:

IF(ACCOUNT-NUM IS <u>NOT</u> NUMERIC AND BILLED-AMOUNT IS NUMERIC <u>OR</u>...

The experience of performing this experiment showed that, while the number of correlated mutant pairs increase as program size grows, the fraction of all mutant pairs that are correlated diminishes. Therefore, the experiment was

extremely time-consuming (in terms of computer time) for large programs. This effect would be expected to intensify for higher order mutation, or larger programs. Thus because of practical constraints, the correlation of mutants cannot be studied further using the method of this experiment.

Higher Order Mutants

It is also possible to look at triples of mutants, or even mutants of higher order. We do not need to carry this too far. The more errors introduced into a program (or from another point of view, the more changes necessary to make a faulty program correct) the more we violate the competent programmer assumption. But we do need some data on multiple mutations, just to assure ourselves that nothing drastic happens as the order of mutation increases. For this experiment 20,000 complex substitution mutants of each of the orders 2, 3, 4, and 5 were generated for each of the six programs. We restrict ourselves to substitutions to avoid the technical difficulties discussed in the random pair experiment. As was stated in the preceeding section, it is not feasible to look at high order correlated mutants. The tuples were checked to make sure that all mutations were applied to different data references. The following table shows the number of mutants that passed the first order test data for each program, and the number that were not equivalent (uncoupled mutants).

Table 6. 20,000 Mutants of Order 2,3,4, and 5 for Each Program

		Program #1	#2	#3	#4	#5	#6
2nd Order Mutants	Number that Pass Test	1	2	5	0	9	5
	Uncoupled Errors (Nonequiv.)	0	0	1	0	0	0
3rd Order Mutants	Number that Pass Test	0	0	0	0	0	0
	Uncoupled Errors (Nonequiv.)	0	0	0	0	0	0
4th Order Mutants	Number that Pass Test	0	0	0	0	0	0
	Uncoupled Errors (Nonequiv.)	0	0	0	0	0	0
5th Order Mutants	Number that Pass Test	0	0	0	0	0	0
	Uncoupled Errors (Nonequiv.)	0	0	0	0	0	0

There are no surprises in this data. Higher order mutants are more easily eliminated. The one uncoupled error is of Type A. The implication of this data is that, at least for the class of potential errors that are

representable as combinations of simple mutations, our experiments on mutant pairs will serve to provide upper bound information on the incidence of uncoupled errors, since higher order mutations are extremely unlikely to be uncoupled.

One other statistic was generated during this case study. For each program and each order of mutation, the average number of statements executed per mutant before the termination of execution (by normal end or error) was calculated.

Table 7. Average Statements Executed Before Failure on Programs with Multiple Order Mutations

Program	2nd Order	3rd Order	4th Order	5th Order
1	30	24	21	19
2	47	27	19	15
3	50	38	31	27
4	124	85	67	59
5	52	35	27	22
6	132	98	74	60

Many software reliability estimates are based on the assumption that the probability of failure in a given time interval of a program is proportional to the number of errors in the program [13]. If that were true, then the expected time to failure of the program would be inversely proportional to the number of errors present. For if T is the time to failure (say in statements executed), and cn is the probability of failure during the execution of any given statement, The the expected time to failure is given by

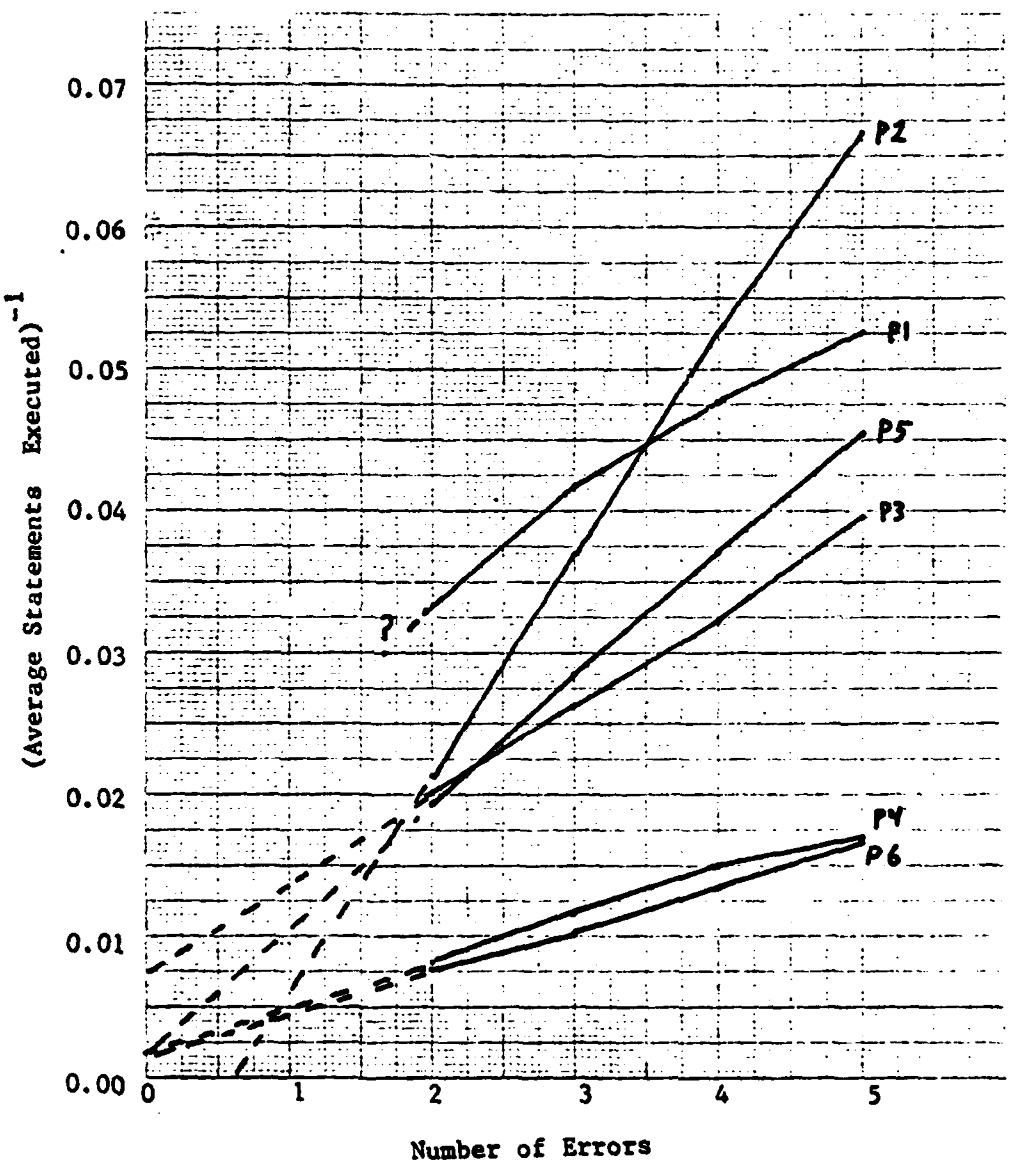


Figure 2. Inverse of Time to Failure vs. Number of Seeded Errors

$$E(T) = \sum_{i=1}^{\infty} (1-cn)^{(i-1)} (cn)(i)$$

which reduces to

$$E(T) = \frac{1}{cn}$$

Table 7 then represents a simulation study of this assumption. As the graph in Figure 2 shows, the assumption is supported quite well. Not only is there apparently a strong linear relationship between 1/Avg(T) and n for each of the programs, but also for all but one of the programs the line segments can be extrapolated backwards to show intercepts near zero. That one program is the smallest and, presumably, the worst simulation of a large software system. This data cannot be interpreted as complete proof of the assumption on the probability of program fialure, however, since the assumption is based on typical "live" input data. The test cases that generated the data were intentionally chosen to be nontypical, in that the test cases were required to execute exception-handling code that would rarely be executed in practice.

Coupling and Complexity

It is possible that some attributes of programs measurable by objective means would have some influence on the strength of coupling. One such attribute to be studied is the structural complexity of programs (measured for

example by the number of branches). One problem with another testing strategy, DD path coverage, is that it may take test data forcing the program down a particular complex path in the program to force the discovery of an error. For example consider the following small program to sort the tuple (A,B,C).

```
L1: if A<B then goto L2;
    T:=A;A:=B;B:=C;
L2: if B<C then goto L3;
    T:=A;A:=C;C:=T;
L3: if B<C then goto L4;
    T:=B;B:=C;C:=T;
L4: stop
```

The program is incorrect. The condition at L2 should be A<C. The input tuples (1,2,3) and (3,2,1) for A,B,and C both give correct results, and force the execution of all DDP's. (1,2,3) takes the TRUE branches at L1, L2, and L3, while (3,2,1) takes the FALSE branches. It is when trying to develop a test case that will cause the execution of the complex path having different results at the last two tests (TRUE at L2 and FALSE at L1, or vice versa), that the error must be discovered. So simply covering all simple path segments may not be sufficient. It is possible that mutation analysis has this same weakness, since mutations are of a highly localized nature. Any weakness would be to a lesser degree, however, since mutation analysis includes DD path coverage as a subcase. To test the relationship of complexity to coupling, we hypothesize that the more branches a program has, the harder it is to test adequately

by mutation analysis. If this is true, the more structurally complex the program, the higher the proportion of uncoupled potential errors we would expect. An experiment to test this hypothesis would match programs for length and number of mutants, but of differing branch-count, and would measure the coupling coefficient defined in Chapter II. If the confidence intervals on the estimates of the coefficients overlap, then we detect no relationship. If they do not, then we have a statistical relationship. If the relationship is found to hold, it would be an argument for simplicity in program structure for programs to be tested by mutation analysis. Currently mutation analysis does not suggest that simplicity is a virtue. For this experiment, "live" data could not be used. Instead, a sequence of small programs was written, all using the same data items and data references, but with an increasing number of branches. The Experiment used 50,000 pairs of mutants for each program. Table 3 shows the number of branches, test case records, mutants, pairs passing the test data, and uncoupled mutants (mutants that pass but are not equivalent) for each program.

Table 8. Comlexity and Coupling

Program	Number of Branches	Number of Records	Number of Mutants	Number that Pass	Number Uncoup-led
C-1	0	1	474	329	0
C-2	1	3	480	153	1
C-3	3	7	492	84	1
C-4	5	12	504	50	3
C-5	7	15	516	18	9

Eleven of the surviving nonequivalent mutants are of Type A, and the other three are of Type B. The large numbers of equivalent mutants in the simple programs are due to "almost useless" statements that were included as places to insert branches without greatly affecting the number of mutants generated.

The effect of adding complexity is very slight, and can be totally accounted for by the type of uncoupled mutants seen in earlier experiments. Hence complexity, at least in terms of branching, is not a hinderence to mutation analysis. Of course these conclusions apply to a very restricted definition of "complexity". When mutation analysis systems become availible for a structured language like Pascal, it will be possible to measure testability and coupling in terms of other structural factors. In particular a comparison of an algorithm coded using GOTO with a comparable algorithm using the more socially acceptable constructs would be interesting.

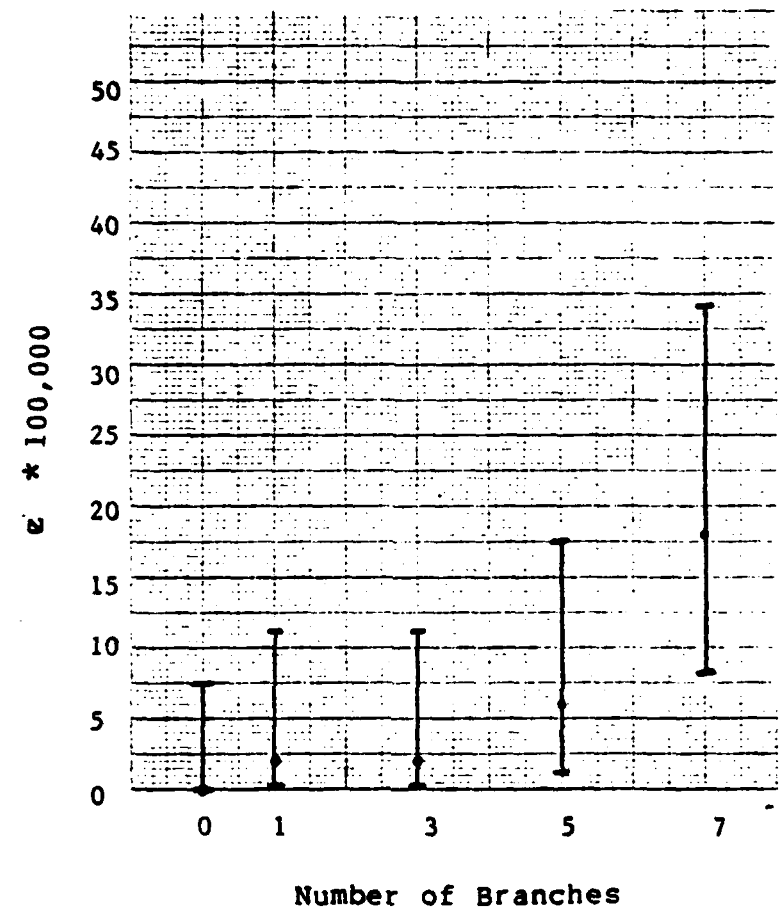


Figure 3. 95% Confidence Intervals on z* 100,000 vs. Number of Branches

CHAPTER V

EQUIVALENCE OF MUTANTS

Human Evaluation of Equivalence

It was stated in Chapter III that it would be possible to detect some equivalent mutants automatically, but not all of them. For that reason we need a mesaure of how accurately humans judge equivalence. An experiment was designed to obtain such a measure under circumstances similar to those under which equivalence judgements would be made in actual testing. Programs 3,4,5,and 5 were used. For each program the sequence of test cases discussed in Chapter III was used to eliminate mutants, but testing was stopped when the number of mutants remaining was approximately twice the number of equivalent mutants. This process eliminated most of the obviously inequivalent mutants. It has been our experience with mutation systems that users rarely examine mutants closely with a view toward detecting equivalences until the set of mutants has been so reduced by testing. From the remaining mutants, for each program a subset of fifty was selected randomly using a pseudo-random number generator. Two subjects were used in the experiment. Both have been involved in the development

of mutation analysis systems, and are competent programmers. Neither had previously been exposed significantly to the programs used in the experiment. Each subject was given the list of mutants and the source listing for each of the programs, and was instructed to mark each mutant "equivalent" or "not equivalent". There was no time limit. The reference answers were prepared by the author in consultation with others.

There are two types of errors that can be made in judging equivalence. The first type is the marking of a non-equivalent mutant as equivalent, and the second is the opposite: marking an equivalent mutant as non-equivalent. The second type is not too serious in the process of mutation analysis, since the mutant remains in the system and may be reconsidered later. The first type is the major problem. When a type 1 error occurs, a non-equivalent mutant which presumably could be valuable in the testing process, and which may directly indicate the presence of an error, is removed prematurely from consideration. Committing a type 1 error increases the likelihood that an erroneous program will be accepted as correct by a practicioner of mutation anlysis. The result of the experiment is shown in Table 9. For each of the four programs, the table shows the number of equivalent and non-equivalent mutants in the sample of fifty mutatns present late in the testing procedure, and the number of

correct identifications, type 1 errors, and type 2 errors for the two subjects.

Table 9. Human Evaluation of Equivalence

Program	# Eq.	# Not	Subject 1			Subject 2		
			Correct	Type 1	Type 2	Correct	Type 1	Type 2
3	20	30	44	0	6	42	2	6
4	21	29	36	2	12	33	6	11
5	20	30	46	0	4	40	5	5
6	13	37	33	16	1	45	1	4

Subject 1 was more variable in accuracy than Subject 2, but overall their results were very similar. Subject 1 identified 79.5% of the mutants correctly. Subject 2 was correct on 80% of the mutants. In measuring type 1 errors, the best computation is probably the total type 1 errors as a percentage of total non-equivalent mutants, since the non-equivalent mutants represent the potential type 1 errors. Subject 1 made type 1 errors on 14.3% of the non-equivalent mutants, and Subject 2 on 11.1%. Similarly, Subject 1 made type 2 errors on 31.5% of the equivalent mutants, and Subject 2 on 35.1% of them.

The measure of type 1 errors may be high enough to reduce confidence in mutation analysis, if it acurately predicted the frequency of such errors in practice. It should be remembered, however, that the subjects were required to choose one mark or the other for each mutant

with the evidence in hand (the source listing), while a tester in practice may postpone the decision pending further thought and testing. Further, the subjects worked in isolation, and were thus denied both helpful consultation and the motivation of accountability for potential errors. These would be important factors in real-life testing situations. On the other hand, the higher error rates for type 2 erors indicate that the subjects were being conservative in their judgements, marking mutants non-equivalent when in doubt.

Pairs of Equivalent Mutants

It might be instructive to look at pairs of mutants that are equivalent as first order mutants. These might be a source of weakness in the mutation approach. The reason is this. An equivalent mutant is a potential error about which the tester is saying "I don't want to bother with this; it isn't important." As single mutants, that may be true, but a pair of equivalent mutants may represent a pair of arbitrary choices made by the programmer, which may not interact properly. From another point of view, if muatations are considered not as errors but as corrections to a buggy program, it may be that the program needs two corrections, neither of which improves the program by itself.

Consider the program fragment

```
P:        A=1
          B=1
           .
           .
           .
          IF A.NE.0 .AND. B.EQ.1  ...
```

Mutant programs P1 with A=1 changed to A=2 and P2 with B=1 changed to B=A might each be equivalent to P, but P12 with <u>both</u> changes might not. If P12 is actually the <u>correct program</u>, then it might be possible for P to pass first order mutation analysis, even though it is incorrect. An experiment aimed at investigating this phenomenon was conducted. For each program, all possible pairs of mutants marked equivalent in the testing process were created and run on the test data. The numbers that were killed were determined. These numbers represent a lower bound on the number of pairs not equivalent to the original program, since the test data is not perfect. For programs 5 and 6, the pairs were randomly sampled due to their great number and to the long run time of the program.

Table 10. Pairs of Equivalent Mutants

Program	Number Equivalent	Number of Pairs Considered	Number Killed by Data	Percent Killed
1	21	208	0	0.00
2	47	1081	4	0.37
3	106	5113	36	0.70
4	95	4283	6	0.14
5	228	5000*	6	0.12**
6	425	5000*	27	0.54***

* random sample
** 95% confidence interval = [0.04 , 0.26]
*** 95% confidence interval = [0.37 , 0.78]

The results show that less than 1% of the pairs of equivalent mutants are determined to be nonequivalent (as pairs) by this test data. These measurements are lower bounds, since stronger test data might distinguish more pairs from the original program. However, the uniformity of the results would tend to raise our confidence that pairs of first order equivalent mutants will not be a major problem for mutation analysis systems.

CHAPTER VI

SUBSETS OF MUTANTS

Random Selection of Mutants

The quadratic growth in the number of mutants of a program is due to the mutant operators of the substitution type. It has been suggested that those operators are actually too strong, and that a fixed small number of substitutions per reference may produce almost the same error-detection power. The reasoning is that the tester "explains" with a test case why the variable X was used, for example, not why Y was not used [1]. Hence random selection of mutants, at least of the substitution types, may be a way to bring the growth of the number of mutants down to the linear range while sacrificing very little power. Table 10 summarizes the results of this study. The columns labeled "survive" indicate the counts of the number of mutants out of the full 100% that survive the specified testing criterion and are not equivalent to the original program.

Table 11. Reduced Power Mutation Analysis

Program	# Mutants at 10%	# Mutants at 100%	Survive "TRAP"	Survive 10% data
1	389	1098	6	0
2	503	2814	906	0
3	1125	6340	129	2
4	1609	7334	97	16
5	1527	7957	407	14
6	4011	28275	789	66

It can be seen that simply generating test data to cover all statements in the program (TRAP) is not very strong, but generating data to eliminate 10% of the mutants is almost as good as using 100% of the mutants. However, the trend as program size increases is not quite what had been expected. As program size increases, 10% mutant selection generates an increasing number of mutations per data reference, and should (intuitively) produce a stronger test. But the strength of the test, measured by the percentage of all mutants eliminated, does not increase with program size, and may actually be decreasing. We may again consider these findings in terms of implications for the design of future mutation analysis systems. Experiments on the coupling effect have already shown that extending mutation from first order to second adds very little testing power. Now it is seen that weakening first order mutation to a subset of itself may decrease the power of the system. This would indicate that first order mutation is not too

strong, but is rather the appropriate level of testing for a mutation analysis system.

Table 12. Test Strength Using 10% of Mutants

Program	lines	Percent Eliminated
1	146	100%
2	163	100%
3	238	99.97%
4	321	99.78%
5	445	99.82%
6	619	99.77%

The test strengths are all very good but studies of this effect with much larger programs are needed to see if our intuitions really are valid.

Efficiency of Mutagenic Operators

A second economy can be gained if it is found that some of the mutant operators provide only error detection capabilities already covered by other mutant operators. In particular, in Cobol, if we do not need the data structure mutants, then we can perform mutations on a machine language internal form (compiled), rather than a higher-level form that must be interpreted.

For a mutatgenic operator (or mutant type) to be useful, it must force the user in some way to produce stronger test data than he would without it. If all of the mutations produced by an operator are extremely unstable (are eliminated by any test data that executes the affected

code), or if all are equivalent, then the operator is not providing useful information and guidance to the tester. Let Nt be the total number of mutants generated by a particular operator, and let Nu be the number that are eliminated on the first execution of the affected code by a test data set, and let Ne be the number equivalent to the original program. Then a measure of the efficiency of the mutagenic operator (for that program and that sequence of test data generation) is given by

$$(Nt - (Nu + Ne)) / Nt$$

Nt and Ne depend only on the program being considered and the operators in use. Nu depends also on the test data generation procedure. It might ber preferable to think of the *in*efficiency

$$(Nu + Ne) / Nt$$

A reasonable procedure for collecting operator efficiency data would be

(1) Select several programs representative of the application space envisioned for testing with a particular mutation system.

(2) Generate test data just strong enough to

execute all statements. (i.e. try to produce weak tests, which cover statements but do as little more as possible.)

(3) Generate test data to eliminate all nonequivalent mutants.

After such measurements have been made on several programs, and preferably even for multiple independent test data generations for each program, a set of efficiency measurements for each mutagenic operator will be obtained. If an operator consistently scores near zero, then the deletion of that operator from the mutation system would be justified. If an operator has a significant efficiency score on any program for any test data generation, then that operator is forcing the tester toward greater test data strength and should be retained.

There are two limitations to this approach. The first is that it does not consider interactions between operators. It may be that two operators each have high efficiencies, but actually have the same effect, i.e. they require the same test data for coverage. In that case one or the other may be necessary, but not both. The efficiency measures will not give us any indication of this. In fact they are giving us just the interaction of the TRAP operator with all of the others, on the assumption that we will always want at

least statement coverage. We could expand the experiment to indicate mutagenic operator dependence on any subset of operators S by replacing step 2 in the procedure with

(2) Generate test data just strong enough to eliminate all of the nonequivalent mutants generated by operators in S.

and by modifying the definition of Nu similarly. Ideally, we would measure the efficiencies of operators relative to all possible subsets, in order to find the minimum subset relative to which no other operators had significant efficiency. Unfortunately, this is not feasible. An approximate operator selection procedure would be to choose the most efficient operator (relative to trap), and call it O1. Next choose the most efficient operator relative to TRAP and O1, and call it O2, and so on. The procedure would terminate when no operator had an efficiency above a practical threshhold.

A second limitation is that the procedure works only for a given class of programs from which we are sampling. Drastically changing language or even the style in which the programs are written would probably affect the choice of efficient mutagenic operators. However if we have a particular population of programs on which we will expend large testing effort, it is possible to "fine tune" the set

of operators for that population of programs, by using only the operators that provide useful testing information.

The results for the single test data generation for the six programs are displayed in Table 12.

Table 13. Mutagenic Operator Efficiencies

Operator	Program					
	1	2	3	4	5	6
Decimal	*	0.96	0.30	0.21	0.33	0.18
Occurs	*	*	*	0.00	*	*
Insert	0.00	0.00	0.01	0.00	0.00	0.00
Fill.Siz	0.00	0.00	0.00	0.00	0.00	0.00
Item Rev	0.05	0.04	0.07	0.00	0.00	0.01
Delete	0.00	0.34	0.00	0.01	0.04	0.03
Go-Perf	*	*	*	0.00	*	0.00
Perf-Go	0.00	0.00	0.00	0.08	0.00	0.00
IF Rev	0.00	0.67	0.00	0.06	0.00	0.00
Stop	0.00	0.00	0.00	0.00	0.00	0.00
Thru	0.00	*	*	0.06	*	0.00
Arith	*	0.75	*	0.04	0.05	*
Compute	*	0.50	0.25	*	0.00	0.00
Parenth.	*	*	0.00	*	0.00	0.00
Round	*	0.44	0.20	0.00	0.11	0.17
Move Rev	0.00	0.00	0.00	0.00	0.04	0.01
Logic	0.07	0.51	0.00	0.13	0.24	0.05
SFS	0.01	0.34	0.03	0.01	0.04	0.02
CFC	0.00	0.25	0.00	0.01	0.10	0.04
CFS	0.00	0.36	0.03	0.01	0.05	0.04
SFC	*	0.18	0.00	0.03	0.09	0.04
C Adjust	0.00	0.50	0.14	0.06	0.22	0.03
Files	0.00	*	*	*	*	0.00

* No mutants of this type generated for this program.

There is a wide variation in efficiencies between programs. This is partly due to the inexact test data selection procedure, and partially due to the inherent differences between programs. The programs use different

language constructs to perform different tasks. A mutagenic operator that focuses attention on one type of construct is most useful in programs that rely heavily on that construct.

The first five operators are of special interest. These data mutations force us into interpretive execution using a run-time symbol table. If they can somehow be avoided, then more efficient compiled execution is possible. The first operator moves the implied decimal point in a numeric item. It is useful primarily in that it forces the tester to provide nonzero values for that variable. The same effect could be achieved by a special mutagenic operator that requires a nonzero value at a data reference in the precedure. FMS.2 provides such an operator called ZPUSH. The second operator alters the OCCURS count in a table description. More investigation of programs using tables is necessary before this operator can safely be deleted, using programs that rely more heavily on table structures. Inserting an extra filler in a record is of little use, as is altering the size of a filler. Reversing two adjacent elementary items within a record is sometimes a useful operation, but probably the same effect is produced by substituting one field for another in the procedure division. A study of the efficiency of item reversal relative to scalar substitution would be useful.

Of the procedural mutations, changing a GO TO to a PERFORM or vice versa usually provides no testing power.

Perhaps most of the testing effect of trying various path alternatives is already achieved by simple statement coverage. Inserting a STOP statement is not helpful because in most programs, files will be left open, an error. STOP insertion thus plays essentially the same role as TRAP. THRU clause alteration, reparenthisization of arithmetic expressions, and the reversal of the direction of a binary MOVE, and changing an I/O reference from one file to another are rarely useful. Probably these mutations too drastic. Errors this large are must be detected by <u>any</u> test data that exercises all of the program. The errors we are looking for after completing basic statement coverage are subtle ones. The major errors have already been ruled out.

A useful but efficient subset of operators for a compiler-based mutation system might therefore be "Delete" (statement deletion), "IF rev" (IF-THEN-ELSE clause reversal), and the substitution operators "Arith" (for arithmetic operator substitution) through "C Adjust" (for constant adjustment) in Table 12.

CHAPTER VII

CONCLUSIONS AND SUGGESTIONS FOR FURTHER STUDY

The results of the experiments reported here basically support mutation analysis as a testing discipline. The experiments on the coupling hypothesis show that test data strong enough to eliminate simple errors is strong enough to eliminate at least 99.977% of random pairs of errors, and 99.79% of correlated pairs. The failings of the coupling effect for higher order errors were too slight to be observed. Program complexity does not seem to create problems for mutation analysis. In all, 1,090,000 complex mutants were considered, and only 45 of them were nonequivalent changes not eliminated by the first order test data. All of the observed failures of the coupling effect were alterations of logical tests, and all but one were either alterations of a comparison operator and one of its operands, or alterations of both operands. We could make a new mutagenic operator: "change a comparison operator and one of its operands", since this would still be only quadratic in program size. Call it CO1. The potential operator CO2: "change both operands of a comparison", is not as attractive, since it would be cubic in program size. However, it is possible that CO2 is coupled to CO1. If an

experiment of the efficiency of C02 relative to C01 (after the fashion of Chapter VI) should support this, then adding one more quadratic operator would correct almost all of the weaknesses of the coupling effect that have been observed in this study.

Less conclusive are the results of the study of the human evaluation of equivalence. It was found that during the necessary step of human judgement of mutant equivalence, errors which weaken the reliability of mutation anlysis may be made with significant frequency. At least until more sensitive studies can be made in a true program testing setting, practicioners of mutation analysis should be cautioned to be very conservative in their marking of equivalence.

Our observation on the efficiencies of the various mutagenic operators indicate that mutation does not inherently limit us to inefficient execution during testing. The operators requiring a run-time symbol table either are not useful or can be simulated by other operators. The operations that were new for this mutation analysis system, affecting input and output, provided no difficulties, at least for the case of read-only and write-only sequential files. Future systems and studies must address more flexible input/output access methods.

In short, the concept of matation analysis has been successfully transferred from Fortran to Cobol, and

experiments performed with the Cobol system provide strong justification for confidence that a program tested with mutation analysis will perform reliably.

APPENDIX A

CMS.1 USERS GUIDE

Allen Acree

July 1, 1979

Document CMS_1.1

INTRODUCTION

The Cobol Mutation System (CMS.1) has been developed at the Georgia Institute of Technology by Allen Acree, Rich DeMillo, Jeanne Hanks, and Fred Sayward. It is based in part on the PIlot Mutation System (PIMS, later renamed FMS.1) for Fortran designed at Yale University, and implemented at Yale University, Georgia Institute of Technology, and the University of California, Berkely.

Program mutation is a method for program testing. The underlying assumption is that programmers produce programs that are, in some sense, nearly correct. The goal of the mutation system is to aid in the selection of good test data by taking advantage of this fact. A mutation of a program P is a program P' that differs from P in only a single minor change, such as substituting one variable for another in an assignment or changing a + to a - in an arithmetic expression. Usually the number of simple mutants of P grows quadratically with the size of P. Naturally, some of these mutations will produce mutant programs that are functionally equivalent to the original, but for the others we should be able to find test data that will distinguish between the original progaram and the mutant.

CMS.1 is designed to take as input a fixed program P, and to automatically produce mutants of it according to a

set of mutagenic operators. The system will then accept test cases from the user, run the original program and all its mutants on it, and tell the user how many mutants have been "killed". (A mutant is killed when it fails by program fault or produces a different output than the original program.) The aim, of course, is to kill all the mutants, or at least to kill enough so that the user is reasonably certain that those remaining are functionally equivalent to the original program and could never be killed. At this point the user has a set of test data that is sufficiently powerful to distinguish between the original program and all its simple (nonequivalent) mutants. According to the coupling hypothesis this test data will also be sufficiently powerful to distinguish between the original program and most other programs "close" to it. (including multiple mutations.) This hypothesis has been proved for certain classes of programs and for certain definitions of "close", and theoretical work continues in this area. Recent experiments with higher order mutants of Fortran and Cobol programs also support this hypothesis.

Thus the user can, with the aid of CMS.1, produce test data that will distinguish between the program used as input and any program "close" to it. Since we assume that the program used as input is close to a correct program, the test data will be sufficient to distinguish between the input program and the correct program, if they are not

equivalent. So the test data will be sufficient to demonstrate program correctness, to a high degree of certainty.

IMPLEMENTATION

The user of CMS.1 provides the name of the file containing the source program. This program should be in the subset of the Cobol language specified later. CMS.1 parses this source program into an internal form suitable for interpretive execution. This internal form is also suitable for "decompilation", and the user can be provided with a decompiled version of any statement. This decompiled statement may not be textually identical to the original source, but it should be equivalent.

The system then produces a file of all mutations of the original program. These are stored, not as complete programs, but rather as short descriptions of how a mutant is to be created. The user is then asked to provide a file or files of test data for his program. These files may be created outside CMS.1 using the editor, or they may be created "on the fly" in CMS.1, with editing capability being restricted to backspace and line delete. However the user choses to provide the input files, CMS.1 interpretively executes the original program on this test data, saving the output. The user may examine the output and decide whether

or not to accept it. If he does, then the test data is run against all enabled mutants, and the results of each are compared to the results of the source. A mutant producing a different result is marked "killed". The user is then presented with a statistical summary. If he wishes, he may also examine more detailed information about the mutants still living. He may also review the test cases accepted so far. Then the cycle repeats until either an error is uncovered in the original program, or the user is satisfied that all remaining mutants are equivalent to the original. A CMS.1 run may be interrupted and continued later, with the system saving all information necessary for the resumption of the run.

In response to the experience of trying to transfer FMS.1 from one environment to another, we have decided to try to do as much as possible to isolate machine dependencies. At the risk of possible inefficiencies, we will concentrate references to file access techniques, character storage, word length, and such machine- and operating system-dependent features in a few small routines. For example, FMS.1 contained 72 random access calls in the DEC Fortran dialect. Each of these had to be rewritten as a PRIMOS call during the transfer procedure. In CMS.1, all random access is through the routines REARAN and WRTRAN. Those two (small) routines are all that need to be modified to interface CMS.1 with a different operating system. For

efficiency, some machine dependency is tolerated in the interpretive execution phase of CMS.1, since this is the most time-consuming phase of the mutation process. However, this dependency is kept to a minimum even here. The buffers used in interpretively executing programs are integer arrays of one or two dimensions. The sizes of the arrays are parameters. We assume in designing these arrays that a single integer consists of at least 16 bits. (i.e. integers are restricted, wherever possible, to a range of +/- 32783.)

NOTES ON THE COBOL PILOT MUTATION SYSTEM

1. We limit ourselves to a simple subset of the language.
2. We limit ourselves to ten sequential input files and ten nonrewindable sequential output files. This should be sufficient for such common applications as making sorted transactions against a sorted master file and producing a transaction report and an updated master file. There is a limit to the amount of storage allocated for each input file and each output file for each test case. The files are "packed" into arrays by replacing each string ot repetitions of a single character (such as a string of blanks) by a single character and a repeat count. This implies that the

user can submit larger test cases (more records) if he can arrange to use such strings whenever possible.

3. Rather than providing for a "predicate subroutine" as in FMS.1 we simply check mutant output against original program output to determine whether they have produced identical output files. Mutants can also be eliminated by run-time faults such as attempting to read an unopened file, data fault, etc. To avoid the infinite loops that some mutations are bound to create, a mutant is eliminated if it executes more than a certain maximum number of statements. Currently this maximum is set to three times the number of statements executed by the original program on the test case.

4. Mutations to be performed:

   1 DECIMAL ALTERATION - Move implied decimal in numeric items one place to the left or right, if possible.

   2 REVERSE TWO-LEVEL TABLE DIMENSIONS

   3 OCCURS CLAUSE ALTERATION - Add or subtract one from an OCCURS clause.

   4 INSERT FILLER - of length one between two items in a record.

   5 FILLER SIZE ALTERATION - Add or subtract one from length.

   6 ELEMENTARY ITEM REVERSAL

   7 FILE REFERENCE ALTERATION

8 STATEMENT DELETION - Replace by null operation.

9 GO TO --> PERFORM

10 PERFORM --> GO TO

11 THEN - ELSE REVERSAL - Negate condition.

12 STOP STATEMENT SUBSTITUTION

13 THRU CLAUSE EXTENSION

14 TRAP STATEMENT REPLACEMENT

15 SUBSTITUTE ARITHMETIC VERB

16 SUBSTITUTE OPERATOR IN COMPUTE

17 PARENTHESIS ALTERATION - Move one parenthesis one place to the left or right

18 ROUNDED ALTERATION - Change ROUNDED to truncation, and vice versa.

19 MOVE REVERSAL - reverse direction of move in simple MOVE A TO B, if the result would be legal in Cobol.

20 LOGICAL OPERATOR REPLACEMENT

21 SCALAR FOR SCALAR REPLACEMENT - Substitute one (non-table) item reference for another, where the result would be legal.

22 CONSTANT FOR CONSTANT REPLACEMENT

23 CONSTANT FOR SCALAR REPLACEMENT

24 SCALAR FOR CONSTANT REPLACEMENT

25 NUMERIC CONSTANT ADJUSTMENT

COBOL SUBSET ACCEPTED BY CMS.1

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

[AUTHOR. comment-entry.]

[INSTALLATION. comment-entry.]

[DATE-WRITTEN. comment-entry.]

[DATE-COMPILED. comment-entry.]

[SECURITY. comment-entry.]

[REMARKS. comment-entry.]

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

[SOURCE-COMPUTER. comment-entry.]

[OBJECT-COMPUTER. comment-entry.]

[SPECIAL-NAMES. [[C01 IS mnemonic-name]

INPUT-OUTPUT SECTION.

FILE-CONTROL.

    [SELECT *file-name* ASSIGN TO {INPUTi | OUTPUTi}...]

    NOTE: 0 <= i <= 9

DATA DIVISION.

FILE SECTION.

```
[FD file-name RECORD CONTAINS integer CHARACTERS
     [LABEL RECORDS ARE { STANDARD| OMITTED }]
     DATA RECORD IS data-name.
     level-number {data-name | FILLER }
     [REDEFINES data-name-2]
     [{ PICTURE | PIC } IS character-string]
     [OCCURS integer TIMES]
     ...
     ...
[WORKING-STORAGE SECTION.
     [77 level entries.]
     [record entries .]...]
```

NOTE: Record entries are the same as in the file section, except VALUE clauses are permitted. Level 88 items (condition names) are not supported. Legal PICTUREs are signed and unsigned numeric, edited numeric, and alphanumeric. The USAGE clause is not supported, and DISPLAY is assumed throughout.

```
PROCEDURE DIVISION.
[paragraph-name.]
     ADD {ident-1 | lit-1} [ident-2 | lit-2]...  { TO |
     GIVING } ident-m
     [ROUNDED] [ON SIZE ERROR imperative-statement] .
     CLOSE filename-1 [filename-2] ...  .
     COMPUTE identifier [ROUNDED] = arithmetic-expression
```

[ON SIZE ERROR imperative]

DIVIDE {ident-1 | lit-1} { INTO | BY } {ident-2 |
lit-2}
[GIVING ident-3] [ROUNDED] [ON SIZE ERROR imperative] .

EXIT.

GO TO paragraph-name

GO TO paragraph-name-1 [[paragraph-name-2] ...
DEPENDING ON identifier].

IF condition {statement-1 | NEXT STATEMENT}
[ELSE {statement-2 | NEXT STATEMENT }] .

NOTE: logical operations AND and OR and comparisons =, NOT GREATER THAN, etc., are permitted. Arithmetic operations within the conditional expression and condition names are not supported. Sign tests and class tests are supported.

MOVE ident-1 TO ident-2 [ident-3] ... .

MULTIPLY {ident-1 | lit-1} BY {ident-2 | lit-2}
[GIVING ident-3] [ROUNDED] [ON SIZE ERROR imperative] .

OPEN [INPUT filename-1 [filename-2]]
[OUTPUT filename-3 [filename-4]]

PERFORM paragraph-name-1 [THRU paragraph-name-2]

PERFORM paragraph-name-1 [THRU paragraph-name-2]
{ident-1 | integer-1} TIMES.

PERFORM paragraph-name-1 [THRU paragraph-name-2]
[VARYING identifier-1 FROM {identifier-2 | literal-1}
BY {identifier-3 | literal-2}] UNTIL condition

READ filename RECORD [INTO identifier]

AT END imperative

STOP RUN.

SUBTRACT {ident-1 | lit-1} [ident-2 | lit-2] ... FROM

{ident-m | lit-m}

[GIVING ident-n] [ROUNDED] [ON SIZE ERROR imperative] .

WRITE record-name [FROM identifier-1]

[AFTER ADVANCING {ident-2 | integer | mnemonic} LINES]

.

THE CMS.1 RUN

The four phases of the CMS.1 run are the ENTRY phase, the PRE-RUN phase, the MUTATION phase, and th POST-RUN phase. The ENTRY phase is executed only when the user first enters the system. Thereafter the PRE-RUN, MUTATION, and POST-RUN phases are exected cyclically.

I. The entry phase.

The session will begin when the user enters the system by logging in and typing

seg run>cpms

If all is well, the system will respond:

WELCOME TO THE COBOL PILOT MUTATION SYSTEM

followed by:

PLEASE ENTER THE NAME OF THE COBOL PROGRAM FILE:

The user should do just that. CMS.1 creates several working files of its own, whose names are variations of the source file name formed by adding suffixes to it. The system checks to see if those working files already exist. If they do, the user can either continue the previous run on that source file where he left off, or he can start over from scratch. Therefore, if the working files already exist, the system asks:

DO YOU WANT TO PURGE WORKING FILES FOR A FRESH RUN ?

If a new run is needed the system begins with the message

PARSING PROGRAM

A syntax error in the source program automatically aborts the CMS.1 run. The user must correct the error and re-enter the system. Errors are reported to the user as a source program line number and the probable cause.

The system then issues the message

SAVING INTERNAL FORM

and asks

WHAT PERCENTAGE OF THE SUBSTITUTION MUTANTS DO YOU WANT TO MAKE?

Since medium to large Cobol programs may generate tens of thousands of mutants, most of which are simple substitutions, the user may want to look initially at only a sampling of the mutants. It has been our experience that eliminating all of the non-equivalent mutants in a 10%

sample gives test data strong enough to eliminate at least 99% of <u>all</u> nonequivalent mutants.

CREATING MUTANT DESCRIPTOR RECORDS

II. The pre-run phase.

In this phase the user supplies test data and turns on mutants. The system asks

DO YOU WANT TO SUBMIT A TEST CASE ?

and the user should respond <u>YES</u> or <u>NO</u>. The system will ask

WHERE IS filename-1 ?

(if there is a SELECT statement for that file)

to which the user should respond <u>HERE</u> or <u><filename></u>

If it is HERE, the user enters the input data directly, ending with the control-C for end of file.

The system then goes through the same procedure for each input file named in a SELECT statement.

At this point the system will execute the program interpretively on the test input. After finishing, the input and output files will be displayed. The user is asked:

IS THIS TEST CASE ACCEPTABLE ?

To which the user should respond <u>YES</u> or <u>NO</u>.

If YES, the test case (input and output, along with the time used, record counts, and a bit map of statements executed) are catalogued for later use with mutant programs. If NO, the test case is purged from memory.

This process of entering test cases iterates until the user states that no more are to be entered at this time.

III. The Mutation Phase

At this time the system will ask

WHAT NEW MUTANT TYPES ARE TO BE CONSIDERED ?

unles all mutant types have already been enabled. The user should respond ALL or NONE or SELECT or should give the numbers of the mutant types to be used next. SELECT causes the system to list each type that has not yes been considered, and then ask for types.

The list of numbers should be terminated with the command STOP. Ranges of types can be specified by TO. For example the reply

14 20 to 25 stop

would enable the TRAP mutants and the data reference substitution mutants.

At this time the test cases will be run against the mutant programs. The time that this takes depends on the number of test cases presented, the length and "density" of the program, and the types of mutants currently being considered. For efficiency, a test case that does not execute a given statement is not executed on any mutant whose mutation is to that statement. The mutant could never be killed if execution never reaches the affected statement. This is the purpose of the bit map saved with the test case.

IV. The Post-Run Phase

After all the test cases have been executed for each mutant still alive, the system will display the statistics of the run, indicating the number of mutants created and the number still alive of each type that has been considered, the percentage of each type killed, and the number of each type marked equivalent. Now the user has a chance to view the mutants still remaining (either all of them, or selected types) or he can send information about the run to an output file for later printing. It is while viewing the live mutants at his terminal that the user has an opportunity to mark the mutants equivalent. After the live mutants, the user has a chance for a similar review of the mutants marked equivalent. He can "unmark" mutants at this time. The user also is able to view or print the test cases at this time. When asked about either the live or equivalent mutants or the test cases, the user may respond YES or NO or OUTPUT. OUTPUT means to send the information to the log file. To end the post-run phase the user types either HALT, ending the session, but saving the temporary files for future resumption, or LOOP sending the system back in a loop to the pre-run phase to enter more test data and/or consider new mutant types.

The user may terminate the session at any time a command is requested by typing KILL, but the state of the system files

after such an abnormal termination is undefined. Continuation of the testing session may not be possible. The user can receive an explanation of his options at many points in the cycle by typing HELP.

CMS.1 AUXILLIARY FILES

CMS.1 creates several files during execution. Some are random access files used for processing the mutants and test cases, and others are needed for the restart capability. When the user provides the name of the file containing the test program, CMS.1 adds suffixes to that name to create names of the auxilliary files. For example if the user provided TEST-PROGRAM-1 as a source program file name, the internal form of the program used by the interpreter and decompiler, would be stored in the file TEST-PROGRAM-1.IF. The test cases would be stored in TEST-PROGRAM-1.TD and TEST-PROGRAM-1.TS, and so fourth. One file deserves special discussion. That is the logfile (TEST-PROGRAM-1.LO in this example). This file contains

(1) A listing of the program, with line numbers.

(2) A statement about the percentage of mutants created.

(3) A summary of test case and mutant transactions, in the order in which they occurred. Whenever a test case

is submitted, a message is logged about that, along with the filenames (or <HERE>) from which the data was obtained, and whether the test case was accepted or rejected. Mutant types are listed as they are enabled.

(4) After each mutation phase the status is written to the file, exactly as it appears on the user's terminal.

(5) An optional listing of the live mutants, provided if the user responds OUTPUT to the question about viewing live mutants.

(6) An optional listing of the test cases, provided if the user responds OUTPUT to that question. A listing of the test cases is strongly recommended. When the test data is displayed on the user's terminal, lines must be truncated to 78 characters. The full lines are placed in the log file.

CMS.1 does not automatically delete these working files after a run is completed. They are retained for possible resumption of testing. It is the responsibility of the user to delete the files when they are no longer needed. The log file is not automatically printed, either. Each run appends to the end of the file where the previous run left off. The user must print the file outside of CMS.1 if he wants a hard copy.

APPENDIX B

CMS.1 INTERNAL SPECIFICATIONS

Allen Acree

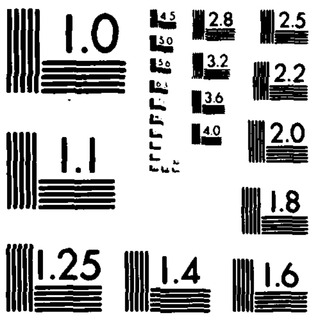
October, 1979

Document CMS_2.2

AD-A091 029   GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
ON MUTATION.(U)
AUG 80 A T ACREE   DAAG29-80-C-0120
UNCLASSIFIED GIT-ICS-80/12   NL

2 2

END
DATE
FILMED
12 80
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

PART I. FILE FORMATS

SOURCE PROGRAM <filename>

The source program is assumed to be in a sequential system file, in the standard COBOL format. That is, columns 1-6 are for the sequence number (and are at this time ignored), column 7 is either blank or contains a hyphen (for the continuation of a non-numeric literal) or an asterisk (for a comment line). Information beyond column 72 is ignored.

INPUT FILE (EXTERNAL)

The input file(s) can also be supplied by the user as standard sequential files. The user only has to tell CMS.1 the name of the file. The alternative is for the user to enter the file directly while he is in CMS.1. When requested, the user should type the file into the terminal, one record per line, just as if he were punching a card deck. The only editing that can be supported in this mode is backspace-erase (control-h), and line-kill (shift-del). The end-of-file is indicated with a control-c. It is of course possible to create some input files outside CMS.1 using whatever tools the user has access to, and to create the others "on the fly" in CMS.1, if the user wishes. Record sizes for input and output files are limited to 150

characters.

TEST FILES (INTERNAL)

The internal test files will contain all test cases that have been created at that time. There are two files containing test information, the test status file, and the test data file.

TEST STATUS FILE <filename>.ts

Each record of the test status file contains 42 words. The first record contains global information.

word	contents
1	1 if INPUT0 is used in the program 0 otherwise.
2 through 20	similar for INPUT1 to INPUT9 and OUTPUT0 to OUTPUT9.
21	The total number of test cases that have been defined.
22	The number of test cases that were defined prior to this pass.
23	pointer to the next record position after the last, for appending.
24 through 42	Not used at this time.

This record will be followed by two records for each test case. The first has the format:

word	contents
1	The starting position of INPUT0 in

<filename>.TD

2	The number of records in INPUTO.
3 through 40	Similar for the other files.
41	The number of statements executed by the original program on this testcase.
42	Not used at this time.

The second record contains a bit map for the statements executed by this test case. If this bit map size (630=42x15) is not adequate, the system parameter TSPRS, which is currently set to 42, may be increased, and the system recompiled. The extra space in the other record types will be wasted.

TEST DATA FILE <filename>.td

The test data file contains the actual test cases, with the input file(s) first, followed by the output file(s) of the original program. These will be in packed format (see PACK and UNPACK), with strings of repeated characters replaced by single characters and repeat counts. The sizes of each file buffer are set by the system parameters IBSZ and OBSZ. In systems where random access files must have fixed record lengths, IBSZ must be equal to OBSZ.

MUTANT RECORD FILE <filename>.mr

The mutant records are stored in binary format, at four integers per mutant record. All records for a particular mutant type are stored contiguously, followed by all records for the next mutant type, etc.

MUTANT STATUS FILE <filename>.ms

The record size for the mutant status file is 16 words. The first section of the file contains headers for each mutant type.

mutant type
on or off ever (initially zero)
on or off this run (" ")
msf record pointer for status block

These may be packed at four headers per 16-word record. All the header blocks remain core-resident during the entire run.

The first record, before these headers, contains a count of the total number of mutants in its first word. The other words are not used.

For each mutant type there is then a status block, of one record.

total mutants for this type
bit map length in words
mrf pointer for the first mutant record of
this type
number of live mutants
number of dead mutants

number killed by trap(*)

number killed by time-out

number killed by data fault

number killed by initialization fault

number killed by I/O fault in OPEN/CLOSE

number killed by attempt to read past EOF

number killed by writing too much

number killed by output too large for buffer

number killed by array subscripts out-of-bounds

number killed by incorrect output

number killed by garbage in the code array

* also includes attempt to execute beyond end of code, such as would happen if a mutation deleted the last STOP RUN statement; and size errors where no SIZE ERROR clause is specified.

The status block will be followed by bit maps.

```
-----------------------
|  live bit map       |
-----------------------

-----------------------
|  dead bit map       |
-----------------------

-----------------------
|  equiv. bit map     |
-----------------------
```

In all of the bit maps, the first bit of each word is not used. The bit maps are of varying lengths, depending on

the program and on the mutant operators. The bit map lengths are rounded up to the nearest whole-record size. The record size for this file is the system parameter MSFRS (currently 16).

NOTE-----We make no provision for keeping information on how each individual mutant was killed. We keep the full matrix of counts of mutant types versus kill mode.

INTERNAL FORM <filename>.if

```
     SYMBOL TABLE      \
                        |
     STATEMENT TABLE    |
                        |
     CODE ARRAY          > binary copies from INFORM
                        |         and HASH
     INIT               |
                        |
     HASH TABLE        /
```

INIT is the initial segment of memory containing literals, PICTUREs, and memory initialization information.

OUTPUT FILE <filename>.lo

This is a sequential file containing information on the run. Its contents are controlled by the user, using the OUTPUT command. Typical contents would be a listing of the source program, the test cases, the status after each pass through the system, and a listing of some or all of the live mutants.

INITIAL.HASH.PACK

The same as HASH-TABLE but containing only the

reserved words and their tokens. This is stored as a packed sequential file. In this case "packed" means that we store a count of null records, followed by a non-null record, followed by a count of null records, etc. until all records (up to the hash table size) are accounted for.

PART II. INTERNAL FORM SPECIFICATIONS

SYMBOL TABLE

The symbol table is an 10xN array of integers. A simple data item (group or elementary) is described by one row in the array. A table item is described in two rows, the second being a dope vector. Some conventions used are that field 1 in each row (record) points to the hash table entry, for the name. If the item has no name (such as a filler or literal), field 1 is zero. Field 2 is always a code for the type of the record. Its value determines the meaning of the other fields.

ROW 1: the program name

Field 1 points to the name, fields 2 to 4 hold integers for the date of last compilation, and the other fields are not used.

ROW 2: INPUTO

field-1 is used for the hash table pointer to the name of the file (as it is known to the program).

field-3 is a pointer to the symbol table entry for the data record.

field-6 is the record length. (field-1 is 0 if there

is no SELECT clause for this device)

ROWS 3 through 21

Like row 2, for INPUT1 to INPUT9, OUTPUT0 to OUTPUT9.

ROW 22 The top-of-page mnemonic for the output files

field-1 points to name in hash, if one has been declared, otherwise it is zero.

DATA ITEMS

field	meaning
1	Index of the identifier in the hash table, so that print name can be recalled. For FILLERS, this is zero.
2	A code for the type of the object. 1 for unsigned numeric identifier 2 for signed numeric identifier 3 for non-numeric identifier 4 for edited numeric item 5 for group item
3	The level number
4	Pointer to the PICTURE string in program memory for edited numeric items. OR the decimal position (from right) for unedited numeric items. OR not used.
5	A pointer to the start of the item in program memory. For an item in a table, this is the constant term in the address calculation.
6	The length of the item, in characters. All items are stored with usage of DISPLAY.
7	The depth of the item in the table structure.

	(0 for scalars, 1 for one-level tables or for rows in two-level tables, 2 for two-level tables entries.)
8	Pointer to VALUE string in program memory.
10	The source program line number on which the item description began

SECOND ROW FOR TABLE ITEMS

field	meaning
2	code = 6
4	the multiplier for the first subscript.
5	the multiplier for the second subscript.
6	The maximum value for subscript-1.
7	The maximum value for subscript-2.
8	The number of OCCURances of the item.

LITERALS DEFINED IN THE PROCEDURE DIVISION

field	meaning
2	code = 7 for numeric literals code = 8 for non-numeric literals code = 10 for the "twiddle" of a numeric literal
4	decimal position, for numeric literal
5	pointer to value in literal pool
6	length

NOTE: SPACES and ZERO (and twiddles of ZERO) have entries of this format which are present by default, even if not used in the program.

PARAGRAPH NAMES

field	meaning
1	pointer to name
2	code = 9
3	statement table index of first statement
4	statement table index of last statement

The symbol table is stored in the same order as the items are encountered in the code. In particular, entries for data items defined in the DATA DIVISION are stored almost line for line as they appear in the source code, with nesting being implicit in the level numbers and the sequence. One deviation from this is the inclusion of dummy FILLER entries of length zero between elementary items. This is to facilitate the mutant operator that inserts fillers to avoid having to change procedure division references.

MEMORY

The first 30 characters of memory are used as a temporary arithmetic register. Following that comes the constant data area. This area includes:

PICture strings - for edited numeric items.

There are 3+N words, where N is the length of the picture string. Word 1 is the length of the string; word 2 is the number of digit positions; and word 3 is the number of digits to the right of the decimal point.

Then follows the picture string, in A1 format. An editing MOVE uses this string to interpretively execute the MOVE instruction.

VALUE literals

for numeric items - word 1 is the number of digits, word 2 is the number of digits in fraction, and words 3 to n+2 are the digits themselves. An operational sign is coded in the last word with the last digit. for nonnumeric items - word 1 is the length N in characters, and words 2 to N+1 are the characters, in A1 format.

Procedure Division literals

Digits or characters only. Since these items have individual symbol table rows, the extra information about length, decimal position, etc, is stored there.

SPACES and ZERO are stored in positions after the arithmetic register in a format that can be referenced either as VALUE or Procedure Division literals, depending on the start pointer.

After the constant area comes the variable area. All data is storage on a USAGE IS DISPLAY basis, one character per word. Since some mutations change the data structure, reallocation between executions is sometimes necessary.

STATEMENT TABLE

The statement table is composed of triples of

integers. field 1: the starting position of an instruction in the code array. When a procedure division statement is mutated, the original code is not modified. Instead, a mutated copy of the instruction is created and appended to the end of the code array. Field 1 is then modified to point to this mutant copy of the instruction.

field 2: The line number of the statement on the source listing.

field 3: A value of 0 means this statement is a continuation in a sentence (no period after previous statement.) A value of 1 means a new sentence. A value greater than 1 means the beginning of an ELSE clause.

INTERNAL FORM OF PROCEDURE DIVISION

Each instruction is preceeded by a word containing the length of that instruction.

meaning	syntax
MOVE	<MOV><n><source><dest-1>...<dest-n>
ADD	<AD><rnd><size><n><op-1>...<op-n>
	(rnd is 0 for truncation, 1 for round) (size is 0 if no SIZE ERROR clause has been specified, and 1 if it has. The SIZE ERROR branch immediately follows the current statement, followed by

	the no error branch.)
ADD-GIVING	<ADG><rnd><size><n><op-1>...<op-n><dest>
SUBTRACT	<SU><rnd><size><n><op-1>...<op-n>
SUB-GIV	<SUG><rnd><size><n><op-1>...<op-n><dest>
MULTIPLY	<MU><rnd><size><op-1><op-2>
MULT-GIV	<MUG><rnd><size><op-1><op-2><dest>
DIVIDE	<DI><rnd><size><op-1><op-2>
DIV-GIV	<DIG><rnd><size><op-1><op-2><dest>
COMPUTE	<CO><rnd><size><ident><arith. exp.>
	note: the arithmetic expression is interpreted by a calculator subroutine.
GO TO	<GO><procedure>
GO TO...DEPEND	<GOD><n><proc-1>...<proc-n><ident>
PERFORM	<PE><procedure><procedure-2>
	(procedure-2 may be null if no THRU clause is specified.)
PERFORM-UNTIL	<PEU><proc-1><proc-2><condition>
PREFORM-VARYING	<PEV><proc-1><proc-2><ident><from><by>
	<REP1><p1-stmt-ptr><p2-code-ptr><cond.>
	REP1 is the iteration control instruction. On return from the PERFORM, the control goes to this instruction. P1-stmt-ptr is a statement table pointer corresponding to the symbol table pointer proc-1. P2-code-ptr is a code pointer for the insertion of the return.
PERFORM-TIMES	<PET><procedure><procedure-2><ident>
	<REP2><count><start><stop>
	Similar to REP1, but count holds the

	value that was in ident when the PET was first executed. Start and stop are statement table pointers for the perform range.
no op	<RET><0>
return	<RET><addr>
	note: each paragraph is ended with a "no op" statememt. When a PERFORM statement is executed, it first changes the no op at the end of its range to a return by inserting the return address (in the statement table) and then transferring to the beginning of the range. When a RETURN is executed, it transfers to the address in the instruction and also changes itself to a no op by changing its address field to 0. No op's are also inserted when NEXT SENTENCE is used or impiled in an IF statement.
IF	<IF><else-stmt-ptr><condition>
	pointer is for transfer if condition is false.
NEGATED IF	<NIF><else-stmt-ptr><condition>
OPEN	<OP><1..20>
	(for which file)
CLOSE	<CL><1..20>
READ	<RE><1..10><from-ident>
WRITE	<WR><1..10><from-ident><advance>
	note: advance is pointer to symtab. Target is either top-of-page mnemonic, an identifier, or a numeric literal.
STOP RUN	<STOP>
TRAP	<TRAP>

NOTES ON THE INTERNAL FORM

1. "identifier","ident", and "id", as well as "op" are pointers to symbol table entries describing identifiers or literals. The symbol table will contain information about type, length, location, etc.

2. Any operand could also be a table reference. In this case, instead of a single integer we would have [op][index-1] or [op][index-1] [index-2]. The interpreter will know from the symbol table entries for op whether 0,1, or 2 indices (subscripts) are needed for a valid reference. Index-1 (and index-2) are also references via the symbol table to simple (unsubscripted) variables or to numeric literals.

3. "procedure" and "proc" are pointers to symbol table entries describing paragraph names. The symbol table will contain pointers to the first and last statements in the paragraph, in the statement table.

MUTANTS

The mutant descriptions are stored in four integers. The first is the mutant type, and the others (not all types use all four integers) are used for auxilliary information, as detailed in PART III.

PART III. DETAILS OF MUTATION PROCESS

MUTANTS

DECIMAL Move implied decimal in numeric items one place to the left or right, if possible.

DIMENS1 Reverse row and column OCCURS counts in a two level table.

DIMENS2 Increment or decrement (by 1) an OCCURS count.

INSERTF Insert a filler with PICTURE X.

ALTERF Alter a filler with PICTURE X(n) to X(n-1) or X(n+1) if possible.

REVERSE Reverse adjacent elementary items in a record.

FILEREF Change a file reference from one input file to another, etc.

DELETE Delete a statement (change it to a NO-OP).

GO-PERF Change a GO TO to a PERFORM, unles the last statement in the paragraph is a stop or transfer of control (in which case it would make no difference).

PERF-GO Change a PERFORM to a GO TO.

THENELS Reverse the "then" and "else" clauses in an IF (negate the condition).

STOPINS Insert a STOP RUN in the program.

THRUEXT Extend the TRHU range of a PERFORM.

TRAP Change a statement to a TRAP, which always fails when

executed. This is for statement coverage information.

ARIVERB Change one arithmetic verb to another.

ARIOPER Change an arithmetic operator in a COMPUTE statement.

PARENTH Alter the parenthesization of an arithmetic expression in a COMPUTE statement.

ROUND Change rounding to truncation, or vice versa.

MOVEREV Reverse the direction of the MOVE in a simple binary move, if such would result in a legal COBOL move.

LOGIC Change a logical comparison to some other comparison.

S-FOR-S Substitute one scalar (unsubscripted) named data reference for another.

C-FOR-C Substitute a constant (numeric or nonnumeric literal) for another.

C-FOR-S Substitute a constant for a scalar.

S-FOR-C Substitute a scalar for a constant.

CONSADJ Increment or decrement a numeric literal by 1 or by 1 whichever is larger.

MUTANT DESCRIPTORS

DATA MUTATIONS

(1) <DECIMAL><sym.tab.loc><+1 | -1><x>

(2) <DIMENS1><sym.tab.loc><x><sym.tab.loc.-2>

for "reverse OCCURS numbers for these two locations". They are assumed to be the

two dimensions for a two-level table.

(3) <DIMENS2><sym.tab.loc><code><x>

where code = 0 for "add 1 to OCCURS"

code = 1 for "subtract 1 from OCCURS"

(4) <INSERTF><symbol table location><x><x>

(5) <ALTERF><sym.tab.loc><+1|-1><x>

(6) <REVERSE><sym.tab.loc.><next.elementary.loc><x>

INPUT/OUTPUT MUTATIONS

(7) <FILEREF><statement><x><new file-code>

CONTROL STRUCTURE MUTATIONS

(8) <DELETE><statement><y><x>

(9) <GO-PERF><statement><x><x>

(10) <PERF-GO><statement><x><x>

(11) <THENELS><statement><x><x>

(12) <STOPINS><statement><x><x>

(13) <THRUEXT><statement><new paragraph limit><x>

(14) <TRAP><statement><x><x>

PROCEDURAL MUTATIONS

(15) <ARIVERB><statement><new operation><x>

to change ADD to SUBTRACT, etc

(16) <ARIOPER><statement><field><new operation>

to change an operation in a COMPUTE.

"field" is the location in code relative to the beginning of the statement. (op code location.)

(17) <PARENTH><statement><from-field><to-field>

(18) <ROUND><statement><x><x>

(19) <MOVEREV><statement><x><x>

(20) <LOGIC><statement><field><new value>

(21) <S-FOR-S><statement><field><new symtab loc.>

(22) <C-FOR-C><statement><field><new loc>

(23) <C-FOR-S><statement><field><new loc>

(24) <S-FOR-C><statement><field><new loc>

(25) <CONSADJ><statement><field><new loc>

Hence the mutants can be stored in a file of 4 x N integers.

APPENDIX C

A CMS.1 Script

The following is a script of a CMS.1 run on a program originally from the Army SIDPERS system. The program has been modified somewhat, mainly in the reduction of the record sizes to make a better CRT display. The program takes as input two files, representing and old backup tape and a new one. The output is a summary of the changes. The input files are assumed to be sorted on a key field. The program has 1195 mutants, of which 21 are easily seen to be equivalent to the original program. Initially ten test cases were generated to eliminate all of the nonequivalent mutants. Subsequently a subset of five test cases was found to be adequate for the task. The entire run took about 10 minutes of clock time, and 2 minutes and 13 seconds of CPU time on the PRIME 400. Note that this is a trace of a terminal session. The output of the testcases is truncated to 78 characters to avoid extra linefeeds. The full output is available on hardcopy to the tester.

```
WELCOME TO THE COBOL PILOT MUTATION SYSTEM
PLEASE ENTER THE NAME OF THE COBOL PROGRAM FILE:>log-changes
DO YOU WANT TO PURGE WORKING FILES FOR A FRESH RUN ?>yes
PARSING PROGRAM
SAVING INTERNAL FORM
WHAT PERCENTAGE OF THE SUBSTITUTION MUTANTS DO YOU WANT TO CREATE?>100
CREATING MUTANT DESCRIPTOR RECORDS
PRE-RUN PHASE
DO YOU WANT TO SUBMIT A TEST CASE ? >program

 PROGRAM LAST COMPILED ON 1 11 80.

      IDENTIFICATION DIVISION.
      PROGRAM-ID. POQAACA.
      AUTHOR. CPT R W MOREHEAD.
      INSTALLATION. HQS USACSC.
      DATE-WRITTEN. OCT 1973.
      REMARKS.
          THIS PROGRAM PRINTS OUT A LIST OF CHANGES IN THE ETF.
          ALL ETF CHANGES WERE PROCESSED PRIOR TO THIS PROGRAM.  THE
          OLD ETF AND THE NEW ETF ARE THE INPUTS.  BUT THERE IS NO
          FURTHER PROCESSING OF THE ETF HERE.  THE ONLY OUTPUT IS A
          LISTING OF THE ADDS, CHANGES, AND DELETES.  THIS PROGRAM IS
          FOR HQ USE ONLY AND HAS NO APPLICATION IN THE FIELD.
           *****************
           MODIFIED FOR TESTING UNDER CPMS BY ALLEN ACREE
           JULY, 1979.
      ENVIRONMENT DIVISION.
      CONFIGURATION SECTION.
      SOURCE-COMPUTER. PRIME.
      OBJECT-COMPUTER. PRIME.
      INPUT-OUTPUT SECTION.
      FILE-CONTROL.
          SELECT OLD-ETF ASSIGN INPUT1.
          SELECT NEW-ETF ASSIGN INPUT2.
          SELECT PRNTR ASSIGN TO OUTPUT1.
      DATA DIVISION.
      FILE SECTION.
      FD  OLD-ETF
          RECORD CONTAINS 80 CHARACTERS
          LABEL RECORDS ARE STANDARD
          DATA RECORD IS OLD-REC.
      01  OLD-REC.
          03  FILLER                          PIC X.
          03  OLD-KEY                         PIC X(12).
          03  FILLER                          PIC X(67).
      FD  NEW-ETF
          RECORD CONTAINS 80 CHARACTERS
```

```
          LABEL RECORDS ARE STANDARD
          DATA RECORD IS NEW-REC.
      01  NEW-REC.
          03  FILLER                          PIC X.
          03  NEW-KEY                         PIC X(12).
          03  FILLER                          PIC X(67).
      FD  PRNTR
          RECORD CONTAINS 40 CHARACTERS
          LABEL RECORDS ARE OMITTED
          DATA RECORD IS PRNT-LINE.
      01  PRNT-LINE                           PIC X(40).
      WORKING-STORAGE SECTION.
      01  PRNT-WORK-AREA.
          03  LINE1                           PIC X(30).
          03  LINE2                           PIC X(30).
          03  LINE3                           PIC X(20).
      01  PRNT-OUT-OLD.
          03  WS-LN-1.
              05  FILLER                      PIC X VALUE SPACE.
              05  FILLER                      PIC XXXX VALUE 'O   '.
              05  LN1                         PIC X(30).
              05  FILLER                      PIC XXX VALUE SPACES.
          03  WS-LN-2.
              05  FILLER                      PIC X VALUE SPACE.
              05  FILLER                      PIC XXXX VALUE 'L   '.
              05  LN2                         PIC X(30).
              05  FILLER                      PIC XXX VALUE SPACES.
          03  WS-LN-3.
              05  FILLER                      PIC X VALUE SPACE.
              05  FILLER                      PIC XXXX VALUE 'D   '.
              05  LN3                         PIC X(20).
              05  FILLER                      PIC XXX  VALUE SPACE.
      01  PRNT-NEW-OUT.
          03  NEW-LN-1.
              05  FILLER                      PIC XXXXX VALUE ' N   '.
              05  N-LN1                       PIC X(30).
              05  FILLER                      PIC XXX VALUE SPACE.
          03  NEW-LN-2.
              05  FILLER                      PIC XXXXX VALUE ' E   '.
              05  N-LN2                       PIC X(30).
              05  FILLER                      PIC XXX VALUE SPACES.
          03  NEW-LN-3.
              05  FILLER                      PIC XXXXX VALUE ' W   '.
              05  N-LN3                       PIC X(20).
              05  FILLER                      PIC XXX VALUE SPACES.
      PROCEDURE DIVISION.
      0100-OPENS.
          OPEN INPUT OLD-ETF NEW-ETF.
          OPEN OUTPUT PRNTR.
      0110-OLD-READ.
          READ OLD-ETF AT END GO TO 0160-OLD-EOF.
      0120-NEW-READ.
          READ NEW-ETF AT END GO TO 0170-NEW-EOF.
      0130-COMPARES.
          IF OLD-KEY = NEW-KEY
              NEXT SENTENCE
          ELSE GO TO 0140-CK-ADD-DEL.
          IF OLD-REC = NEW-REC
              GO TO 0110-OLD-READ.
          MOVE OLD-REC TO PRNT-WORK-AREA.
          PERFORM 0210-OLD-WRT THRU 0210-EXIT.
          MOVE NEW-REC TO PRNT-WORK-AREA.
          PERFORM 0200-NW-WRT THRU 0200-EXIT.
```

```
        GO TO 0110-OLD-READ.
    0140-CK-ADD-DEL.
        IF OLD-KEY > NEW-KEY
            MOVE NEW-REC TO PRNT-WORK-AREA
            PERFORM 0200-NW-WRT THRU 0200-EXIT
            GO TO 0120-NEW-READ
        ELSE GO TO 0150-CK-ADD-DEL.
    0150-CK-ADD-DEL.
        MOVE OLD-REC TO PRNT-WORK-AREA.
        PERFORM 0210-OLD-WRT THRU 0210-EXIT.
        READ OLD-ETF AT END
            MOVE NEW-REC TO PRNT-WORK-AREA
            PERFORM 0200-NW-WRT THRU 0200-EXIT
            GO TO 0160-OLD-EOF.
        GO TO 0130-COMPARES.
    0160-OLD-EOF.
        READ NEW-ETF AT END GO TO 0180-EOJ.
        MOVE NEW-REC TO PRNT-WORK-AREA.
        PERFORM 0200-NW-WRT THRU 0200-EXIT.
        GO TO 0160-OLD-EOF.
    0170-NEW-EOF.
        MOVE OLD-REC TO PRNT-WORK-AREA.
        PERFORM 0210-OLD-WRT THRU 0210-EXIT.
        READ OLD-ETF AT END GO TO 0180-EOJ.
        GO TO 0170-NEW-EOF.
    0180-EOJ.
        CLOSE OLD-ETF NEW-ETF PRNTR.
        STOP RUN.
    0200-NW-WRT.
        MOVE LINE1 TO N-LN1.
        MOVE LINE2 TO N-LN2.
        MOVE LINE3 TO N-LN3.
        WRITE PRNT-LINE FROM NEW-LN-1 AFTER ADVANCING 2.
        WRITE PRNT-LINE FROM NEW-LN-2 AFTER ADVANCING 1.
        WRITE PRNT-LINE FROM NEW-LN-3 AFTER ADVANCING 1.
    0200-EXIT.
        EXIT.
    0210-OLD-WRT.
        MOVE LINE1 TO LN1.
        MOVE LINE2 TO LN2.
        MOVE LINE3 TO LN3.
        WRITE PRNT-LINE FROM WS-LN-1 AFTER ADVANCING 2.
        WRITE PRNT-LINE FROM WS-LN-2 AFTER ADVANCING 1.
        WRITE PRNT-LINE FROM WS-LN-3 AFTER ADVANCING 1.
    0210-EXIT.
        EXIT.
>yes
```

A test case for this program is a pair of input files. In CMS.1 these may be created outside the system and referenced by name, or may be entered "on the fly".

```
 WHERE IS OLD-ETF?
>lc9
 WHERE IS NEW-ETF?
>lc6
 OLD-ETF PROVIDED TO THE PROGRAM

 I1234567890I2IIIIIIIIII0JJJJJJJJJKKKKKKKKKKLLLLLLLLLLNNNNNNNNNNBBBBBBBBBBGGGGG
 J234567890123YYYYYYYYYYGGGGGGGGGGFFFFFFFFFF0DDDDDDDDDDSSSSSSSSSSXXXXXXXXXXEEEEE

 NEW-ETF PROVIDED TO THE PROGRAM
```

```
I123456789012000000000000000000000000000000000000000000000000000000000000000000
J234567890123YYYYYYYYYYGGGGGGGGGGFFFFFFFFFFDDDDDDDDDDSSSSSSSSSSXXXXXXXXXXEEEEE
 345678901234UUUUUUUUUUHHHHHHHHHHGGGGGGGGGGDDDDDDDDDDSSSSSSSSSSEEEEEEEEEEAAAAA

PRNTR AS WRITTEN BY THE PROGRAM

 O    I123456789012IIIIIIIIII0JJJJJJ
 L    JJJKKKKKKKKKKLLLLLLLLLLNNNNNNN
 D    NNNBBBBBBBBBBGGGGGGG

 N    I12345678901200000000000000000
 E    000000000000000000000000000000
 W    00000000000000000000

 O    J234567890123YYYYYYYYYYGGGGGGG
 L    GGGFFFFFFFFFF0DDDDDDDDDSSSSSSS
 D    SSSXXXXXXXXXXEEEEEEE

 N    J234567890123YYYYYYYYYYGGGGGGG
 E    GGGFFFFFFFFFFDDDDDDDDDDSSSSSSS
 W    SSSXXXXXXXXXXEEEEEEE

 N     345678901234UUUUUUUUUUHHHHHHH
 E    HHHGGGGGGGGGGDDDDDDDDDDSSSSSSS
 W    SSSEEEEEEEEEEAAAAAAA

 THE PROGRAM TOOK  84 STEPS
IS THIS TEST CASE ACCEPTABLE ? >yes
DO YOU WANT TO SUBMIT A TEST CASE ? >no
MUTATION PHASE
WHAT NEW MUTANT TYPES ARE TO BE CONSIDERED ?  >select

ENTER THE NUMBERS OF THE MUTANT TYPES YOU WANT TO TURN ON AT THIS TIME.

      4       ****  INSERT FILLER TYPE  ****
      5       ****  FILLER SIZE ALTERATION TYPE  ****
      6       ****  ELEMENTARY ITEM REVERSAL TYPE  ****
      7       ****  FILE REFERENCE ALTERATION TYPE  ****
      8       ****  STATEMENT DELETION TYPE  ****
     10       ****  PERFORM --> GO TO TYPE  ****
     11       ****  THEN - ELSE REVERSAL TYPE  ****
     12       ****  STOP STATEMENT SUBSTITUTION TYPE  ****
     13       ****  THRU CLAUSE EXTENSION TYPE  ****
     14       ****  TRAP STATEMENT REPLACEMENT TYPE  ****
     19       ****  MOVE REVERSAL TYPE  ****
     20       ****  LOGICAL OPERATOR REPLACEMENT TYPE  ****
     21       ****  SCALAR FOR SCALAR REPLACEMENT  ****
     22       ****  CONSTANT FOR CONSTANT REPLACEMENT  ****
     23       ****  CONSTANT FOR SCALAR REPLACEMENT  ****
     25       ****  CONSTANT ADJUSTMENT  ****

TYPES ? >4 to 14 stop
--- TESTCASE      1 ---
      250
      284 CONSIDERED        224 KILLED        60 REMAIN
MUTANT STATUS

TYPE        TOTAL     LIVE      PCT   EQUIV
INSERT         41        7    82.93     0
FILLSZ         38       14    63.16     0
ITEMRV         21        0   100.00     0
FILES           5        1    80.00     0
DELETE         54       13    75.93     0
```

```
PER GO           7          2   71.43     0
IF REV           3          1   66.67     0
STOP            53         10   81.13     0
THRU             8          2   75.00     0
TRAP            54         10   81.48     0

TOTALS
               284         60   78.87     0
DO YOU WANT TO SEE THE LIVE MUTANTS?>no
DO YOU WANT TO SEE THE EQUIVALENT MUTANTS?>no
WOULD YOU LIKE TO SEE THE TEST CASES?>no
LOOP OR HALT ? >loop
PRE-RUN PHASE
DO YOU WANT TO SUBMIT A TEST CASE ? >yes
 WHERE IS OLD-ETF?
>lc15
 WHERE IS NEW-ETF?
>lc5
 OLD-ETF PROVIDED TO THE PROGRAM

 0000000000012IIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLNNNNNNNNNNBBBBBBBBBBGGGGG
 I123456789012IIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLNNNNNNNNNNBBBBBBBBBBGGGGG
 J234567890123YYYYYYYYYYGGGGGGGGGGFFFFFFFFFFDDDDDDDDDDSSSSSSSSSSXXXXXXXXXXEEEEE

 NEW-ETF PROVIDED TO THE PROGRAM

 I123456789012IIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLNNNNNNNNNNBBBBBBBBBBGGGGG
 J234567890123YYYYYYYYYYGGGGGGGGGGFFFFFFFFFFDDDDDDDDDDSSSSSSSSSSXXXXXXXXXXEEEEE

 PRNTR AS WRITTEN BY THE PROGRAM

  O   0000000000012IIIIIIIIIIJJJJJJJ
  L   JJJKKKKKKKKKKLLLLLLLLLLNNNNNNN
  D   NNNBBBBBBBBBBGGGGGGG

 THE PROGRAM TOOK  44 STEPS
IS THIS TEST CASE ACCEPTABLE ? >yes
DO YOU WANT TO SUBMIT A TEST CASE ? >yes
 WHERE IS OLD-ETF?
>lc14
 WHERE IS NEW-ETF?
>lc5
 OLD-ETF PROVIDED TO THE PROGRAM

 I123456789012IIIIIIIIIIKJJJJJJJJJKKKKKKKKKKLLLLLLLLLLNNNNNNNNNNBBBBBBBBBBGGGGG
 J234567890123YYYYYYYYYYGGGGGGGGGGFFFFFFFFFFDDDDDDDDDDSSSSSSSSSSXXXXXXXXXXEEEEE

 NEW-ETF PROVIDED TO THE PROGRAM

 I123456789012IIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLNNNNNNNNNNBBBBBBBBBBGGGGG
 J234567890123YYYYYYYYYYGGGGGGGGGGFFFFFFFFFFDDDDDDDDDDSSSSSSSSSSXXXXXXXXXXEEEEE

 PRNTR AS WRITTEN BY THE PROGRAM

  O   I123456789012IIIIIIIIIIKJJJJJJ
  L   JJJKKKKKKKKKKLLLLLLLLLLNNNNNNN
  D   NNNBBBBBBBBBBGGGGGGG

  N   I123456789012IIIIIIIIIIJJJJJJJ
  E   JJJKKKKKKKKKKLLLLLLLLLLNNNNNNN
  W   NNNBBBBBBBBBBGGGGGGG

 THE PROGRAM TOOK  48 STEPS
```

```
IS THIS TEST CASE ACCEPTABLE ? >yes
DO YOU WANT TO SUBMIT A TEST CASE ? >yes
 WHERE IS OLD-ETF?
>lcll
 WHERE IS NEW-ETF?
>lcl
 OLD-ETF PROVIDED TO THE PROGRAM

 0000000000000000000000000000000000000000

 NEW-ETF PROVIDED TO THE PROGRAM

 I123456789012IIIIIIIIIJJJJJJJJJKKKKKKKKKKLLLLLLLLLNNNNNNNNNNBBBBBBBBBBGGGGG
 J234567890123YYYYYYYYYYGGGGGGGGGGFFFFFFFFFDDDDDDDDDDSSSSSSSSSSXXXXXXXXXXEEEEE
  345678901234UUUUUUUUUUHHHHHHHHHHGGGGGGGGGGDDDDDDDDDDSSSSSSSSSSEEEEEEEEEEAAAAA

 PRNTR AS WRITTEN BY THE PROGRAM

  O   000000000000000000000000000
  L   0000000000000
  D

  N   I123456789012IIIIIIIIIJJJJJJJ
  E   JJJKKKKKKKKKKLLLLLLLLLLNNNNNNN
  W   NNNBBBBBBBBBBGGGGGGG

  N   J234567890123YYYYYYYYYYGGGGGGG
  E   GGGFFFFFFFFFFDDDDDDDDDDSSSSSSS
  W   SSSXXXXXXXXXXEEEEEEE

  N    345678901234UUUUUUUUUUHHHHHHH
  E   HHHGGGGGGGGGGDDDDDDDDDDSSSSSSS
  W   SSSEEEEEEEEEEAAAAAAA

 THE PROGRAM TOOK  64 STEPS
IS THIS TEST CASE ACCEPTABLE ? >yes
DO YOU WANT TO SUBMIT A TEST CASE ? >yes
 WHERE IS OLD-ETF?
>lcl
 WHERE IS NEW-ETF?
>lcll
 OLD-ETF PROVIDED TO THE PROGRAM

 I123456789012IIIIIIIIIJJJJJJJJJKKKKKKKKKKLLLLLLLLLNNNNNNNNNNBBBBBBBBBBGGGGG
 J234567890123YYYYYYYYYYGGGGGGGGGGFFFFFFFFFDDDDDDDDDDSSSSSSSSSSXXXXXXXXXXEEEEE
  345678901234UUUUUUUUUUHHHHHHHHHHGGGGGGGGGGDDDDDDDDDDSSSSSSSSSSEEEEEEEEEEAAAAA

 NEW-ETF PROVIDED TO THE PROGRAM

 0000000000000000000000000000000000000000

 PRNTR AS WRITTEN BY THE PROGRAM

  N   000000000000000000000000000
  E   0000000000000
  W

  O   I123456789012IIIIIIIIIJJJJJJJ
  L   JJJKKKKKKKKKKLLLLLLLLLLNNNNNNN
  D   NNNBBBBBBBBBBGGGGGGG

  O   J234567890123YYYYYYYYYYGGGGGGG
  L   GGGFFFFFFFFFFDDDDDDDDDDSSSSSSS
```

```
  D   SSSXXXXXXXXXXEEEEEEE

  O    345678901234UUUUUUUUUUHHHHHHH
  L   HHHGGGGGGGGGGDDDDDDDDDDSSSSSSS
  D   SSSEEEEEEEEEEEAAAAAAA

 THE PROGRAM TOOK  64 STEPS
IS THIS TEST CASE ACCEPTABLE ? >yes
DO YOU WANT TO SUBMIT A TEST CASE ? >no
MUTATION PHASE
WHAT NEW MUTANT TYPES ARE TO BE CONSIDERED ?  >all
--- TESTCASE     1 ---
        250
        500
        750
        814 CONSIDERED       640 KILLED       174 REMAIN
--- TESTCASE     2 ---
        234 CONSIDERED        82 KILLED       152 REMAIN
--- TESTCASE     3 ---
        152 CONSIDERED         1 KILLED       151 REMAIN
--- TESTCASE     4 ---
        151 CONSIDERED        61 KILLED        90 REMAIN
--- TESTCASE     5 ---
         90 CONSIDERED        69 KILLED        21 REMAIN
MUTANT STATUS

TYPE       TOTAL    LIVE      PCT   EQUIV
INSERT        41          3   92.68      0
FILLSZ        38         12   68.42      0
ITEMRV        21          0  100.00      0
FILES          5          0  100.00      0
DELETE        54          1   98.15      0
PER GO         7          0  100.00      0
IF REV         3          0  100.00      0
STOP          53          0  100.00      0
THRU           8          0  100.00      0
TRAP          54          0  100.00      0
MOVE R        13          0  100.00      0
LOGIC         15          1   93.33      0
SUBSFS       704          4   99.43      0
SUBCFC        12          0  100.00      0
SUBCFS        58          0  100.00      0
C ADJ         12          0  100.00      0

TOTALS
            1098         21   98.09      0
DO YOU WANT TO SEE THE LIVE MUTANTS?>yes
   THE LIVE MUTANTS

FOR EACH MUTANT : HIT RETURN TO CONTINUE.  TYPE 'STOP' TO STOP.
TYPE 'EQUIV' TO JUDGE THE MUTANT EQUIVALENT.

 ****  INSERT FILLER TYPE  ****

THERE ARE          3  MUTANTS OF THIS TYPE LEFT.
DO YOU WANT TO SEE THEM?>yes
 A FILLER OF LENGTH ONE HAS BEEN INSERTED AFTER
 THE ITEM WHICH STARTS ON LINE 52
 ITS LEVEL NUMBER IS 3

>
 A FILLER OF LENGTH ONE HAS BEEN INSERTED AFTER
 THE ITEM WHICH STARTS ON LINE 53
```

```
 ITS LEVEL NUMBER IS 3

>
 A FILLER OF LENGTH ONE HAS BEEN INSERTED AFTER
 THE ITEM WHICH STARTS ON LINE 69
 ITS LEVEL NUMBER IS 3

>

 ****  FILLER SIZE ALTERATION TYPE  ****

THERE ARE         12  MUTANTS OF THIS TYPE LEFT.
DO YOU WANT TO SEE THEM?>yes
 THE FILLER ON LINE 58  HAS HAD ITS SIZE  DECREMENTED  BY ONE.

>
 THE FILLER ON LINE 58  HAS HAD ITS SIZE  INCREMENTED  BY ONE.

>
 THE FILLER ON LINE 63  HAS HAD ITS SIZE  DECREMENTED  BY ONE.

>
 THE FILLER ON LINE 63  HAS HAD ITS SIZE  INCREMENTED  BY ONE.

>
 THE FILLER ON LINE 68  HAS HAD ITS SIZE  DECREMENTED  BY ONE.

>
 THE FILLER ON LINE 68  HAS HAD ITS SIZE  INCREMENTED  BY ONE.

>
 THE FILLER ON LINE 73  HAS HAD ITS SIZE  DECREMENTED  BY ONE.

>
 THE FILLER ON LINE 73  HAS HAD ITS SIZE  INCREMENTED  BY ONE.

>
 THE FILLER ON LINE 77  HAS HAD ITS SIZE  DECREMENTED  BY ONE.

>
 THE FILLER ON LINE 77  HAS HAD ITS SIZE  INCREMENTED  BY ONE.

>
 THE FILLER ON LINE 81  HAS HAD ITS SIZE  DECREMENTED  BY ONE.

>
 THE FILLER ON LINE 81  HAS HAD ITS SIZE  INCREMENTED  BY ONE.

>

 ****  STATEMENT DELETION TYPE  ****

THERE ARE         1  MUTANTS OF THIS TYPE LEFT.
DO YOU WANT TO SEE THEM?>yes
 ON LINE 106  THE STATEMENT:
              GO TO 0150-CK-ADD-DEL
 HAS BEEN DELETED.

>

 ****  LOGICAL OPERATOR REPLACEMENT TYPE  ****

THERE ARE         1  MUTANTS OF THIS TYPE LEFT.
```

```
DO YOU WANT TO SEE THEM?>yes
 ON LINE 102  THE STATEMENT:
             IF OLD-KEY  > NEW-KEY
 HAS BEEN CHANGED TO:
             IF OLD-KEY NOT < NEW-KEY

>

 ****  SCALAR FOR SCALAR REPLACEMENT  ****

THERE ARE         4  MUTANTS OF THIS TYPE LEFT.
DO YOU WANT TO SEE THEM?>yes
 ON LINE 129  THE STATEMENT:
             MOVE LINE1 TO N-LN1
 HAS BEEN CHANGED TO:
             MOVE NEW-REC TO N-LN1

>
 ON LINE 129  THE STATEMENT:
             MOVE LINE1 TO N-LN1
 HAS BEEN CHANGED TO:
             MOVE PRNT-WORK-AREA TO N-LN1

>
 ON LINE 138  THE STATEMENT:
             MOVE LINE1 TO LN1
 HAS BEEN CHANGED TO:
             MOVE OLD-REC TO LN1

>
 ON LINE 138  THE STATEMENT:
             MOVE LINE1 TO LN1
 HAS BEEN CHANGED TO:
             MOVE PRNT-WORK-AREA TO LN1

>
DO YOU WANT TO SEE THE EQUIVALENT MUTANTS?>no
WOULD YOU LIKE TO SEE THE TEST CASES?>no
LOOP OR HALT ? >halt

**** STOP
```

APPENDIX D

An FMS.1 Script on a CMS.1 Module

MUTATION ON MUTATION

This is a report of an experience in using the program mutation methodology on a production software module, namely, a subroutine in another mutation system. The subject subroutine is NXTLIV from the Cobol pilot mutation system (CMS.1) being developed by the author at Georgia Tech. Since CMS.1 is written in Fortran, NXTLIV was run on the pilot mutation system for Fortran (FMS.1) which was developed at Yale University and later transferred to Georgia Tech.

Previous experiments of this kind have taken a routine believed to be correct, and performing mutation analysis on it to (1) increase confidence in the module's correctness, and (2) demonstrate that first order mutation analysis is feasible for real programs. The current study differs primarily in that the routine was known to contain at least one error. The error had resisted the usual debugging techniques (selective trace, etc.) Hence FMS.1 was being used in this instance not as a test data evaluator, but as a tool for systematic debugging, and, perhaps just as importantly, as a convenient test bed for a subroutine extracted from its normal environment.

The routine NXTLIV takes as input the identifying number of a mutant of a given type, and returns the number of the next live mutant, as indicated by bit maps of the live mutants. The bit maps are in general too large to fit in an internal array, so they are "paged" from a random access disk file as needed. Similar maps are kept of the dead mutants and the mutants judged to be equivalent.

The original program:

```
      SUBROUTINE NXTLIV(MTYPE,MUTNO)
C  FIND THE NEXT LIVE MUTANT AFTER THE MUTNOth OF TYPE MTYPE
C  RETURN THIS VALUE IN MUTNO.
C  A VALUE OF ZERO RETURNED MEANS NO MUTANTS OF THAT TYPE REMAIN ALIVE
      NOLIST
$INSERT ICS057>CPMS.COMPAR>SYSTEM.PAR
$INSERT ICS057>CPMS.COMPAR>MACHINE.SIZES.PAR
$INSERT ICS057>CPMS.COMPAR>FILENM.COM
$INSERT ICS057>CPMS.COMPAR>TSTDAT.COM
$INSERT ICS057>CPMS.COMPAR>MSBUF.COM
      LIST
      INTEGER MTYPE,MUTNO
      INTEGER I,J,K,L,WORD,BIT
      LOGICAL ERR
C      CALL TIMER1(33)
C  ASSUME THAT THE RECORD CONTAINING THE LIVE BIT MAPS FOR
C  MUTNO IS ALREADY PRESENT, UNLESS MUTNO=0
      K=BPW-1
C  CHECK TO SEE IF WE ARE AT THE END OF A PHYSICAL RECORD
      IF(MUTNO.EQ.0)GOTO 1
      IF(MOD(MUTNO,K*MSFRS).EQ.0)GOTO 24
      GOTO 10
1     CALL REARAN(MSFILE,LIVBUF,MSFRS,LIVPTR,ERR)
      IF(ERR)CALL ABORT('(NXTLIV) ERROR IN MUTANT STATUS FILE',36)
      CALL REARAN(MSFILE,EQUBUF,MSFRS,EQUPTR,ERR)
      IF(ERR)CALL ABORT('(NXTLIV) ERROR IN MUTANT STATUS FILE',36)
      CALL REARAN(MSFILE,DEDBUF,MSFRS,DEDPTR,ERR)
      IF(ERR)CALL ABORT('(NXTLIV) ERROR IN MUTANT STATUS FILE',36)
      CHANGD=.FALSE.
      WORD=1
      BIT=2
      GOTO 20
10    WORD=MOD((MUTNO)/(K),MSFRS)+1
      BIT=MOD(MUTNO,K)+2
20    DO 22 J=WORD,MSFRS
      L=LIVBUF(J)
      IF(L.NE.0)GOTO 23
```

```
      MUTNO=MUTNO+K
      IF(MUTNO.GT.MCT)GOTO 40
      GOTO 22
23    DO 21 I=BIT,BPW
      MUTNO=MUTNO+1
      IF(MUTNO.GT.MCT)GOTO 40
      IF(AND(L,2**(BPW-I)).NE.0)GOTO 30
21    CONTINUE
      BIT=2
22    CONTINUE
24    IF(.NOT.CHANGD)GOTO 25
C  SAVE OLD RECORDS
      CALL WRTRAN(MSFILE,LIVBUF,MSFRS,LIVPTR,ERR)
      CALL WRTRAN(MSFILE,EQUBUF,MSFRS,EQUPTR,ERR)
      CALL WRTRAN(MSFILE,DEDBUF,MSFRS,DEDPTR,ERR)
C  NEED TO GET NEXT RECORDS
25    LIVPTR=LIVPTR+MSFRS
      EQUPTR=EQUPTR+MSFRS
      DEDPTR=DEDPTR+MSFRS
      GOTO 1
30    GOTO 9999
40    MUTNO=0
      IF(.NOT.CHANGD)GOTO 9999
C  SAVE OLD RECORDS
      CALL WRTRAN(MSFILE,LIVBUF,MSFRS,LIVPTR,ERR)
      CALL WRTRAN(MSFILE,EQUBUF,MSFRS,EQUPTR,ERR)
      CALL WRTRAN(MSFILE,DEDBUF,MSFRS,DEDPTR,ERR)
9999  CONTINUE
C       CALL TIMER2
      RETURN
      END
```

FMS.1 accepts a limited subset of Fortran, and thus the program could not be tested directly as it came from CMS.1.

(1) PARAMETER statements are not accepted, so the parameters BPW (bits per word), MSFRS (mutant status file record size) which come from the $INSERT blocks were systematically replaced by convenient constants, 4 and 4.

(2) CALL statements are not supported. The random I/O routines are simulated by arrays to be read from and written to. The two TIMER routines are not essential and can be ignored.

(3) The functions MOD and AND are not available and had to be simulated.

(4) Type LOGICAL is not available and had to be simulated by INTEGER.

The modified program:

```
      SUBROUTINE NXTLIV(MUTNO,MCT,LIVBUF,NLB,LLB,CHANGD)
C  FIND THE NEXT LIVE MUTANT AFTER THE MUTNOth OF TYPE MTYPE
C  RETURN THIS VALUE IN MUTNO.
C  A VALUE OF ZERO RETURNED MEANS NO MUTANTS OF THAT TYPE REMAIN ALIVE
      INTEGER MUTNO,TEMP
      INTEGER I,J,L,WORD,BIT
      INTEGER MCT,LIVBUF(4),LLB(4),NLB(4),CHANGD
C  ASSUME THAT THE RECORD CONTAINING THE LIVE BIT MAPS FOR
C  MUTNO IS ALREADY PRESENT, UNLESS MUTNO=0
C  CHECK TO SEE IF WE ARE AT THE END OF A PHYSICAL RECORD
      IF(MUTNO.EQ.0)GOTO 1
CCCC        IF(MOD(MUTNO,K*MSFRS).EQ.0)GOTO 24
      IF((MUTNO/12)*12.EQ.MUTNO)GOTO 24
      GOTO 10
1     DO 111 I = 1,4
111   LIVBUF(I)=NLB(I)
```

```
CCCC1      CALL REARAN(MSFILE,LIVBUF,MSFRS,LIVPTR,ERR)
CCCC       IF(ERR)CALL ABORT('(NXTLIV) ERROR IN MUTANT STATUS FILE',36)
CCCC       CALL REARAN(MSFILE,EQUBUF,MSFRS,EQUPTR,ERR)
CCCC       IF(ERR)CALL ABORT('(NXTLIV) ERROR IN MUTANT STATUS FILE',36)
CCCC       CALL REARAN(MSFILE,DEDBUF,MSFRS,DEDPTR,ERR)
CCCC       IF(ERR)CALL ABORT('(NXTLIV) ERROR IN MUTANT STATUS FILE',36)
CCCC       CHANGD=.FALSE.
      CHANGD=0
      WORD=1
      BIT=2
      GOTO 20
CCCC10     WORD=MOD((MUTNO)/(K),MSFRS)+1
10    WORD=((MUTNO/3)-4*((MUTNO/3)/4)) + 1
CCCC       BIT=MOD(MUTNO,K)+2
      BIT=MUTNO-3*(MUTNO/3) + 2
20    DO 22 J=WORD,4
      L=LIVBUF(J)
      IF(L.NE.0)GOTO 23
      MUTNO=MUTNO+3
      IF(MUTNO.GT.MCT)GOTO 40
      GOTO 22
23    DO 21 I=BIT,4
      MUTNO=MUTNO+1
      IF(MUTNO.GT.MCT)GOTO 40
CCCC       IF(AND(L,2**(BPW-I)).NE.0)GOTO 30
      TEMP=L/(2**(4-I))
      IF(TEMP.NE.(TEMP/2)*2) GOTO 30
21    CONTINUE
      BIT=2
22    CONTINUE
CCCC24     IF(.NOT.CHANGD)GOTO 25
24    IF(CHANGD.EQ.0)GOTO 25
C  SAVE OLD RECORDS
CCCC       CALL WRTRAN(MSFILE,LIVBUF,MSFRS,LIVPTR,ERR)
CCCC       CALL WRTRAN(MSFILE,EQUBUF,MSFRS,EQUPTR,ERR)
CCCC       CALL WRTRAN(MSFILE,DEDBUF,MSFRS,DEDPTR,ERR)
      DO 241 I=1,4
241   LLB(I)=LIVBUF(I)
C  NEED TO GET NEXT RECORDS
CCCC25     LIVPTR=LIVPTR+MSFRS
CCCC       EQUPTR=EQUPTR+MSFRS
CCCC       DEDPTR=DEDPTR+MSFRS
CCCC       GOTO 1
25    GOTO 1
30    GOTO 9999
40    MUTNO=0
CCCC       IF(.NOT.CHANGD)GOTO 9999
      IF(CHANGD.EQ.0) GOTO 9999
C  SAVE OLD RECORDS
CCCC       CALL WRTRAN(MSFILE,LIVBUF,MSFRS,LIVPTR,ERR)
CCCC       CALL WRTRAN(MSFILE,EQUBUF,MSFRS,EQUPTR,ERR)
CCCC       CALL WRTRAN(MSFILE,DEDBUF,MSFRS,DEDPTR,ERR)
      DO 291 I=1,4
291   LLB(I)=LIVBUF(I)
9999  CONTINUE
      RETURN
      END
```

A trace of the initial FMS.1 run on this routine appears below, with commentary in lower case.

```
OK, SEG RUN>PIMS
  PRE-RUN PHASE
```

```
 ALL INPUT MUST BE IN UPPER CASE
 ENTER THE RAW PROGRAM FILE NAME
NXTLIV
 DO YOU WANT TO PURGE WORKING FILES
 FOR A FRESH START?
 TYPE A YES OR NO   ****
YES
 CATEGORIZE FORMAL PARAMETER MUTNO
IO
 CATEGORIZE FORMAL PARAMETER MCT
IN
 CATEGORIZE FORMAL PARAMETER LIVBUF
IO
 CATEGORIZE FORMAL PARAMETER NLB
IN
 CATEGORIZE FORMAL PARAMETER LLB
IO
 CATEGORIZE FORMAL PARAMETER CHANGD
IO
 IS MUTANT CORRECTNESS DEPENDENT ON A PREDICATE SUBROUTINE?
 TYPE A YES OR NO   ****
NO
 HOW MANY TEST CASES ARE TO BE SPECIFIED?
1
 SPECIFY TEST CASE      1
 ENTER VALUES FOR
 MUTNO ,MCT    ,CHANGD,
0 6 0
```

a value of "0" for mutno on input means that this is a new mutant type, and a new record is required. MCT is the total number of mutants of the current type.

```
      ENTER     4 VALUES FOR ARRAY LIVBUF
7 7 0 0
 ENTER     4 VALUES FOR ARRAY NLB
0 0 0 0
```

NLB is the next live buffer. In this case is should be transferred to LIVBUF for use immediately.

```
      ENTER     4 VALUES FOR ARRAY LLB
0 0 0 0
  TEST CASE NUMBER    1
 PARAMETERS ON INPUT
 MUTNO    =         0
```

a minor bug in the Georgia Tech version of FMS.1 prevents the input on the first testcase from being echoed.

```
      PARAMETERS ON OUTPUT
 MUTNO    =         0
 LIVBUF   (   1)=         0
 LIVBUF   (   2)=         0
 LIVBUF   (   3)=         0
 LIVBUF   (   4)=         0
 LLB      (   1)=         0
 LLB      (   2)=         0
 LLB      (   3)=         0
 LLB      (   4)=         0
 CHANGD   =         0
   THE RAW PROGRAM TOOK    41 STEPS TO EXECUTE THIS TEST CASE
 HIT RETURN TO CONTINUE
```

mutno=0 on output means that the end of the live mutant map for this type has been reached.

```
 PLEASE VERIFY THAT DATA IS CORRECT
 TYPE A YES OR NO   ****
YES
  WHAT NEW TYPES OF MUTANTS ARE TO BE CONSIDERED ?
PAN
```

this stands for "path analysis". The mutant operator replaces statements with a <trap> statement which always causes the mutant to fail if the statement is executed.

```
  WHAT NEW TYPES OF MUTANTS ARE TO BE CONSIDERED ?
NONE
  MUTATION PHASE
  POST RUN PHASE
 NUMBER OF TEST CASES =     1        NUMBER OF MUTANTS =      44
 NUMBER OF LIVE MUTANTS =      23   PCT OF ELIMINATED MUTANTS =   47.73

 MUTANT TYPES AND LIVE MUTANTS PROFILES
 TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE*
 PAN   44   23*

 MUTANT ELIMINATION METHOD PROFILE
 METHOD    COUNT* METHOD    COUNT* METHOD    COUNT* METHOD    COUNT*
 TIMED-OUT     0* REF UNDVAR    0* SUBSCR RNG    0* ZERO DIV      0*
 ARTH FAULT    0* RDONLY VAR    0* TRAP STMT.   21* WRONG ANS     0*
 EQUIV         0*
 POST RUN RESULTS
MUTANTS
     MUTANT NUMBER          2

  16        IF((MUTNO/12)*12.EQ.MUTNO)GOTO 24
          STATEMENT HAS BEEN CHANGED TO
  16        TRAP
HIT RETURN TO CONTINUE, TYPE STOP TO FINISH
TYPE EQUIV TO KILL MUTANT

     MUTANT NUMBER          3

  17        GOTO 10
          STATEMENT HAS BEEN CHANGED TO
  17        TRAP
HIT RETURN TO CONTINUE, TYPE STOP TO FINISH
TYPE EQUIV TO KILL MUTANT

     MUTANT NUMBER         10

  32  10    WORD=((MUTNO/3)-4*((MUTNO/3)/4)) + 1
          STATEMENT HAS BEEN CHANGED TO
  32  10    TRAP
HIT RETURN TO CONTINUE, TYPE STOP TO FINISH
TYPE EQUIV TO KILL MUTANT

     MUTANT NUMBER         11

  34        BIT=MUTNO-3*(MUTNO/3) + 2
          STATEMENT HAS BEEN CHANGED TO
  34        TRAP
HIT RETURN TO CONTINUE, TYPE STOP TO FINISH
TYPE EQUIV TO KILL MUTANT
STOP
 TYPE NEXT COMMAND
LOOP
  PRE-RUN PHASE
 SAVING OUTPUT FILE ON BAKOUT
```

```
 HIT RETURN TO CONTINUE

 HOW MANY NEW TEST CASES FOR THIS RUN?
1
 SPECIFY TEST CASE      2
 ENTER VALUES FOR
 MUTNO ,MCT   ,CHANGD,
1 6 0
 ENTER      4 VALUES FOR ARRAY LIVBUF
7 7 0 0
 ENTER      4 VALUES FOR ARRAY NLB
0 0 0 0
 ENTER      4 VALUES FOR ARRAY LLB
0 0 0 0
  TEST CASE NUMBER     2
 PARAMETERS ON INPUT
 MUTNO    =          1
 MCT      =          6
 LIVBUF   (   1)=           7
 LIVBUF   (   2)=           7
 LIVBUF   (   3)=           0
 LIVBUF   (   4)=           0
 NLB      (   1)=           0
 NLB      (   2)=           0
 NLB      (   3)=           0
 NLB      (   4)=           0
 LLB      (   1)=           0
 LLB      (   2)=           0
 LLB      (   3)=           0
 HIT RETURN TO CONTINUE

 LLB      (   4)=           0
 CHANGD   =          0
 PARAMETERS ON OUTPUT
 MUTNO    =          2
 LIVBUF   (   1)=           7
 LIVBUF   (   2)=           7
 LIVBUF   (   3)=           0
 LIVBUF   (   4)=           0
 LLB      (   1)=           0
 LLB      (   2)=           0
 LLB      (   3)=           0
 LLB      (   4)=           0
 CHANGD   =          0
   THE RAW PROGRAM TOOK     16 STEPS TO EXECUTE THIS TEST CASE
 HIT RETURN TO CONTINUE

 PLEASE VERIFY THAT DATA IS CORRECT
 TYPE A YES OR NO    ****
YES
  WHAT NEW TYPES OF MUTANTS ARE TO BE CONSIDERED ?
NONE
 REVIEW PREVIOUS RUN RESULTS
GO
  MUTATION PHASE
  POST RUN PHASE
 NUMBER OF TEST CASES =     2       NUMBER OF MUTANTS =      44
 NUMBER OF LIVE MUTANTS =      11   PCT OF ELIMINATED MUTANTS =    75.00

 MUTANT TYPES AND LIVE MUTANTS PROFILES
 TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE*
 PAN   44   11*

 MUTANT ELIMINATION METHOD PROFILE
```

```
 METHOD   COUNT* METHOD   COUNT* METHOD   COUNT* METHOD   COUNT*
 TIMED-OUT    0* REF UNDVAR   0* SUBSCR RNG   0* ZERO DIV     0*
 ARTH FAULT   0* RDONLY VAR   0* TRAP STMT   33* WRONG ANS    0*
 EQUIV        0*
 POST RUN RESULTS
LOOP
  PRE-RUN PHASE
 SAVING OUTPUT FILE ON BAKOUT
 HIT RETURN TO CONTINUE

 HOW MANY NEW TEST CASES FOR THIS RUN?
1
 SPECIFY TEST CASE      3
 ENTER VALUES FOR
 MUTNO ,MCT   ,CHANGD,
10 20 1
 ENTER     4 VALUES FOR ARRAY LIVBUF
1 3 0 0
 ENTER     4 VALUES FOR ARRAY NLB
7 7 0 0
 ENTER     4 VALUES FOR ARRAY LLB
99 99 99 99
  TEST CASE NUMBER     3
 PARAMETERS ON INPUT
 MUTNO    =       10
 MCT      =       20
 LIVBUF   (   1)=          1
 LIVBUF   (   2)=          3
 LIVBUF   (   3)=          0
 LIVBUF   (   4)=          0
 NLB      (   1)=          7
 NLB      (   2)=          7
 NLB      (   3)=          0
 NLB      (   4)=          0
 LLB      (   1)=         99
 LLB      (   2)=         99
 LLB      (   3)=         99
 HIT RETURN TO CONTINUE

 LLB      (   4)=         99
 CHANGD   =        1
 PARAMETERS ON OUTPUT
 MUTNO    =       14
 LIVBUF   (   1)=          7
 LIVBUF   (   2)=          7
 LIVBUF   (   3)=          0
 LIVBUF   (   4)=          0
 LLB      (   1)=          1
 LLB      (   2)=          3
 LLB      (   3)=          0
 LLB      (   4)=          0
 CHANGD   =        0
  THE RAW PROGRAM TOOK     56 STEPS TO EXECUTE THIS TEST CASE
 HIT RETURN TO CONTINUE
```

An error has been detected. The correct output for MUTNO is 13 instead of 14. The error resulted from choosing a starting point in the middle of a word of zero bits. NXTLIV ordinarily loops through the bits of each word looking for the next "1" bit, but as an efficiency measure, a whole word is compared to zero before entering the loop. If all bits are off, MUTNO is incremented by the word length, and the next word is accessed. The Correct algorithm would increment MUTNO only by the number of bits left to be examined in the word. The only way this could make a difference in the original program is for NXTLIV to be called

in such a way as to stop at a "1" bit in the middle of the word, and then have the system turn off the bit by reason of mutant failure of equivalence (outside NXTLIV), and then have NXTLIV called again for the next mutant to be considered. This situation is rare enough to frustrate haphazard debugging attempts, but common enough to cause irritation in a production-sized run.

The correction is to replace

```
MUTNO = MUTNO + 3
(MUTNO = MUTNO + K in the original)
by
MUTNO = MUTNO + (3-(BIT-2))
(MUTNO = MUTNO + (K-(BIT-2)) in the original).
```

After correcting this error, the program was re-entered to FMS.1 and the testing cycle started over.

```
      OK, SEG RUN>PIMS
  PRE-RUN PHASE
 ALL INPUT MUST BE IN UPPER CASE
 ENTER THE RAW PROGRAM FILE NAME
NXTLIV
 DO YOU WANT TO PURGE WORKING FILES
 FOR A FRESH START?
 TYPE A YES OR NO    ****
YES
 CATEGORIZE FORMAL PARAMETER MUTNO
IO

 etc.

 HOW MANY TEST CASES ARE TO BE SPECIFIED?
1
 SPECIFY TEST CASE      1
 ENTER VALUES FOR
 MUTNO ,MCT   ,CHANGD,
0 5 1
```

and so fourth. Test cases were entered and executed correctly until all of the path analysis mutants were eliminated.

```
  POST RUN PHASE
 NUMBER OF TEST CASES =    8      NUMBER OF MUTANTS =     44
 NUMBER OF LIVE MUTANTS =  0      PCT OF ELIMINATED MUTANTS =  100.00

 MUTANT TYPES AND LIVE MUTANTS PROFILES
 TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE*
 PAN   44    0*

 MUTANT ELIMINATION METHOD PROFILE
 METHOD    COUNT* METHOD    COUNT* METHOD    COUNT* METHOD    COUNT*
 TIMED-OUT      0* REF UNDVAR    0* SUBSCR RNG     0* ZERO DIV        0*
 ARTH FAULT     0* RDONLY VAR    0* TRAP STMT     44* WRONG ANS       0*
 EQUIV          0*
```

There is no claim made that this number of test cases is an any way minimal. Some killed only one mutant.

```
 POST RUN RESULTS
LOOP
  PRE-RUN PHASE
 SAVING OUTPUT FILE ON BAKOUT
 HIT RETURN TO CONTINUE

 HOW MANY NEW TEST CASES FOR THIS RUN?
0
  WHAT NEW TYPES OF MUTANTS ARE TO BE CONSIDERED ?
SELECT
```

```
 FOR EACH CHOICE, TYPES YES, NO OR FINISH
        ARRAY LIMIT DEFAULT INSERTION   *
YES
        2-DIM ARRAY LIMIT PERMUTATION   *
NO
        CONSTANT REPLACEMENT            *
YES
        SCALAR VARIABLE REPLACEMENT     *
NO
        SCALAR VAR FOR CONSTANT REPLMT  *
NO
        CONSTANT FOR SCALAR VAR REPLMT  *
NO
        COMPARABLE ARRAY NAME REPLMT    *
YES
        CONST FOR ARRAY REF REPLACEMNT  *
NO
        SCALAR VAR FOR ARR REF REPLMT   *
YES
        ARRAY REF FOR CONST REPLACEMNT  *
NO
        ARR REF FOR SCALAR VAR REPLMT   *
NO
        2-DIM ARRAY REF INDEX PERMUTE   *
NO
        SCALAR VAR INIT INSERTION       *
NO
        ARITHMETIC OPERATOR REPLACEMNT  *
YES
        RELATIONAL OPERATOR REPLACEMNT  *
YES
        LOGICAL CONNECTOR REPLACEMENT   *
NO
        ARITHMETIC PRECEDENCE PERMUTE   *
NO
        LOGICAL PRECEDENCE PERMUTATION  *
NO
        GOTO LABEL REPLACEMENT          *
YES
        CONTINUE STATEMENT INSERTION    *
NO
        CONTINUE STATEMENT DELETION     *
NO
        INNER DO-LOOP DECOUPLING        *
NO
        DO-LOOP INDEX ALTERATION        *
YES
        RETURN STATEMENT INSERTION      *
YES
 THESE MUTANT TYPES WERE ALREADY ON:
 PAN
  WHAT NEW TYPES OF MUTANTS ARE TO BE CONSIDERED ?
NONE
 REVIEW PREVIOUS RUN RESULTS
GO
  MUTATION PHASE
  POST RUN PHASE
 NUMBER OF TEST CASES =     8        NUMBER OF MUTANTS =     399
 NUMBER OF LIVE MUTANTS =       25   PCT OF ELIMINATED MUTANTS =    93.73

 MUTANT TYPES AND LIVE MUTANTS PROFILES
 TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE*
 ALD    3    0* CAR   14    0* SFA   63    0* AOR   84    2*
 ROR   40   10* GLR  108    9* PAN   44    0* RSR   43    4*
```

```
 MUTANT ELIMINATION METHOD PROFILE
 METHOD   COUNT* METHOD   COUNT* METHOD   COUNT* METHOD   COUNT*
 TIMED-OUT   31* REF UNDVAR  16* SUBSCR RNG  19* ZERO DIV     5*
 ARTH FAULT   0* RDONLY VAR   0* TRAP STMT   44* WRONG ANS  259*
 EQUIV        0*
 POST RUN RESULTS
HALT
```

later...

```
OK, SEG RUN>PIMS
  PRE-RUN PHASE
 ALL INPUT MUST BE IN UPPER CASE
 ENTER THE RAW PROGRAM FILE NAME
NXTLIV
 DO YOU WANT TO PURGE WORKING FILES
 FOR A FRESH START?
 TYPE A YES OR NO    ****
NO
```

after entering several test cases, the situation was as shown:

```
  MUTATION PHASE
  POST RUN PHASE
 NUMBER OF TEST CASES =    11       NUMBER OF MUTANTS =     399
 NUMBER OF LIVE MUTANTS =       9  PCT OF ELIMINATED MUTANTS =   97.74

 MUTANT TYPES AND LIVE MUTANTS PROFILES
 TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE*
 ALD    3    0* CAR   14    0* SFA   63    0* AOR   84    0*
 ROR   40    1* GLR  108    7* PAN   44    0* RSR   43    1*

 MUTANT ELIMINATION METHOD PROFILE
 METHOD   COUNT* METHOD   COUNT* METHOD   COUNT* METHOD   COUNT*
 TIMED-OUT   31* REF UNDVAR  16* SUBSCR RNG  20* ZERO DIV     5*
 ARTH FAULT   0* RDONLY VAR   0* TRAP STMT   44* WRONG ANS  262*
 EQUIV       12*
 POST RUN RESULTS
LOOP
  PRE-RUN PHASE
 SAVING OUTPUT FILE ON BAKOUT
 HIT RETURN TO CONTINUE
```

It was decided to leave those nine alone, and consider all mutants, including the multitude of substitution mutants.

```
 HOW MANY NEW TEST CASES FOR THIS RUN?
0
  WHAT NEW TYPES OF MUTANTS ARE TO BE CONSIDERED ?
ALL
 THESE MUTANT TYPES WERE ALREADY ON:
 ALD CRP CAR SFA AOR ROR GLR PAN DIA RSR
 REVIEW PREVIOUS RUN RESULTS
GO
  MUTATION PHASE
  POST RUN PHASE
 NUMBER OF TEST CASES =    11       NUMBER OF MUTANTS =    1514
 NUMBER OF LIVE MUTANTS =      50  PCT OF ELIMINATED MUTANTS =   96.70

 MUTANT TYPES AND LIVE MUTANTS PROFILES
 TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE*
 ALD    3    0* SVR  368   14* SPC  306   12* CPS  180    6*
 CAR   14    0* CFA   24    0* SFA   63    0* APC  104    1*
```

```
AFS  128    4* AOR   84     0* ROR   40     1* GLR  108     7*
PAN   44    0* CSI    3     3* CSD    2     1* RSR   43     1*

MUTANT ELIMINATION METHOD PROFILE
METHOD    COUNT* METHOD    COUNT* METHOD     COUNT* METHOD   COUNT*
TIMED-OUT    45* REF UNDVAR 481* SUBSCR RNG   98* ZERO DIV     25*
ARTH FAULT    0* RDONLY VAR   0* TRAP STMT    44* WRONG ANS   759*
EQUIV        12*
POST RUN RESULTS
```

A cycle of (1) look at a few live mutants
(2) generate test data to kill those mutants
(3) execute mutants on test data
(4) look at more mutants

was followed several times until the mutant was encountered

```
    MUTANT NUMBER        689

  45        BIT=2
          STATEMENT HAS BEEN CHANGED TO
  45        I=2
```

The following data was entered to try to eliminate this mutant. It involved starting in the middle of a word, and having to go into the next word to find the next on bit.

```
 SPECIFY TEST CASE    15
 ENTER VALUES FOR
 MUTNO ,MCT   ,CHANGD,
5 20 0
 ENTER     4 VALUES FOR ARRAY LIVBUF
0 0 1 0
 ENTER     4 VALUES FOR ARRAY NLB
1 1 1 1
 ENTER     4 VALUES FOR ARRAY LLB
99 99 99 99
  TEST CASE NUMBER   15
 PARAMETERS ON INPUT
 MUTNO    =         5
 MCT      =        20
 LIVBUF   (   1)=          0
 LIVBUF   (   2)=          0
 LIVBUF   (   3)=          1
 LIVBUF   (   4)=          0
 NLB      (   1)=          1
 NLB      (   2)=          1
 NLB      (   3)=          1
 NLB      (   4)=          1
 LLB      (   1)=         99
 LLB      (   2)=         99
 LLB      (   3)=         99
 HIT RETURN TO CONTINUE

 LLB      (   4)=         99
 CHANGD   =         0
 PARAMETERS ON OUTPUT
 MUTNO    =         7
 LIVBUF   (   1)=          0
 LIVBUF   (   2)=          0
 LIVBUF   (   3)=          1
 LIVBUF   (   4)=          0
 LLB      (   1)=         99
```

```
 LLB      (  2)=         99
 LLB      (  3)=         99
 LLB      (  4)=         99
 CHANGD  =         0
   THE RAW PROGRAM TOOK    23 STEPS TO EXECUTE THIS TEST CASE
 HIT RETURN TO CONTINUE

 PLEASE VERIFY THAT DATA IS CORRECT
 TYPE A YES OR NO   ****
KILL
 ABORTING RUN

**** STOP 77777
```

The answer is wrong. Another error in the program has been found. Again it is related to the test for an entire word of zeros. If we start in the middle of a word of zeros, the BIT pointer is not being reset to 2 to begin searching the next word. The correction that is needed is to replace

```
      BIT=2
  22  CONTINUE
```

by

```
  22  BIT=2
```

It is interesting to note that another mutant further down in the list does exactly that -- remove the continue statement at the end of a DO loop and put the label on the next-to-last statement. The error was discovered before it absolutely had to be, but it would have been discovered eventually in any case.

```
OK, SEG RUN>PIMS
  PRE-RUN PHASE
 ALL INPUT MUST BE IN UPPER CASE
 ENTER THE RAW PROGRAM FILE NAME
NXTLIV
 DO YOU WANT TO PURGE WORKING FILES
 FOR A FRESH START?
 TYPE A YES OR NO   ****
YES
 CATEGORIZE FORMAL PARAMETER MUTNO
IO
 CATEGORIZE FORMAL PARAMETER MCT
IN
 CATEGORIZE FORMAL PARAMETER LIVBUF
IO
 CATEGORIZE FORMAL PARAMETER NLB
IN
 CATEGORIZE FORMAL PARAMETER LLB
IO
 CATEGORIZE FORMAL PARAMETER CHANGD
IO
 IS MUTANT CORRECTNESS DEPENDENT ON A PREDICATE SUBROUTINE?
 TYPE A YES OR NO   ****
NO
 HOW MANY TEST CASES ARE TO BE SPECIFIED?
15
 SPECIFY TEST CASE     1
 ENTER VALUES FOR
 MUTNO ,MCT   ,CHANGD,
FILE TESTNXT
```

The 15 test cases already generated were run against all mutants on the latest version of the program. These test cases had been saved on a file rather than entered by hand during the run.

```
 MUTATION PHASE
 POST RUN PHASE
NUMBER OF TEST CASES =   15       NUMBER OF MUTANTS =   1580
NUMBER OF LIVE MUTANTS =     52  PCT OF ELIMINATED MUTANTS =   96.71

MUTANT TYPES AND LIVE MUTANTS PROFILES
TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE*
ALD    3    0* CRP   68    5* SVR  368    4* SPC  306   10*
CPS  180    6* CAR   14    0* CPA   24    0* SPA   63    0*
AFC  104    0* AFS  128    2* AOR   84    0* ROR   40    8*
GLR  108    9* PAN   43    0* CSI    4    3* CSD    1    1*
RSR   42    4*

MUTANT ELIMINATION METHOD PROFILE
METHOD   COUNT* METHOD   COUNT* METHOD   COUNT* METHOD   COUNT*
TIMED-OUT   47* REF UNDVAR 483* SUBSCR RNG 111* ZERO DIV    26*
ARTH FAULT   0* RDONLY VAR   0* TRAP STMT   43* WRONG ANS  818*
EQUIV        0*
POST RUN RESULTS
```

The cycle of killing a few mutants at a time was entered again, and some mutants were judged to be equivalent along the way. One principal source of equivalent mutants was the troublesome test for a word of zeros. Its only purpose is to save the effort of looking through the word bit by bit. If the condition in the test is replace by any condition that is identically .TRUE., the program runs a bit longer sometimes, but gets the same result. An example of this is:

```
    MUTANT NUMBER        813

  34        IF(L.NE.0)GOTO 23
          STATEMENT HAS BEEN CHANGED TO
  34        IF( 12.NE.0) GOTO  23
```

Another source of equivalent mutants is the occurrence of extra labels. For example it is easy to see that GOTO 25 can always be replaced with GOTO 1. At some statements in the program a variable is guaranteed to have a particular value. This generates equivalent mutants such as

```
    MUTANT NUMBER        694

  52         DO 241 I=1,4
          STATEMENT HAS BEEN CHANGED TO
  52        DO 241 I=CHANGD,4
```

In all, 37 mutants were judged to be equivalent, and the rest were eliminated by test cases on which the program performed correctly.

One equivalent mutant actually turned out to be an improvement (albeit a slight one) on the original program.

```
    MUTANT NUMBER       1362

  36         IF(MUTNO.GT.MCT)GOTO 40
          STATEMENT HAS BEEN CHANGED TO
  36        IF( MUTNO.GE.MCT) GOTO  40
```

```
MUTANT STATUS AFTER THIS RUN
NUMBER OF TEST CASES =   24      NUMBER OF MUTANTS =   1580
NUMBER OF LIVE MUTANTS =      0  PCT OF ELIMINATED MUTANTS =  100.00

MUTANT TYPES AND LIVE MUTANTS PROFILES
TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE* TYPE MUT LIVE*
ALD    3    0* CRP   68    0* SVR  368    0* SFC  306    0*
CFS  180    0* CAR   14    0* CFA   24    0* SFA   63    0*
AFC  104    0* AFS  128    0* AOR   84    0* ROR   40    0*
GLR  108    0* PAN   43    0* CSI    4    0* CSD    1    0*
RSR   42    0*

MUTANT ELIMINATION METHOD PROFILE
METHOD    COUNT* METHOD    COUNT* METHOD    COUNT* METHOD    COUNT*
TIMED-OUT   51* REF UNDVAR 483* SUBSCR RNG 113* ZERO DIV    26*
ARTH FAULT   0* RDONLY VAR   0* TRAP STMT   43* WRONG ANS  827*
EQUIV       37*
 POST RUN RESULTS
HALT
```

Previous experience has never found a program that has passes mutant analysis that still contained an error. The current program will be a good test of the generality of that experience, since this routine is expected to continue in service for some time. It should be noted that not all of the original routine has been tested by mutation, and no claims are made for the untested portions. But if mutation is valid, the central logic of the routine should now be correct.

APPENDIX E

Statistical Background

STATISTICAL BACKGROUND

Analysis of Variance

In many experimental settings, several factors are thought to have some possible relationship to a response variable which can be measured. Generally a linear model is used.

$$X = E + aA + bB + \ldots$$

Where X = the measured response variable

A = a controlled factor

a = an unknown constant

B = another factor

b = B's constant

etc.

and E = an "error" term; a random variable for variation not accounted for by any of the controlled factors. Some of the factors being considered may be interactions of other factors.

Analysis of variance is a test of each of the hypotheses:

a=0, b=0, ...

Suppose A is controlled to take on just two values, say 0 and 1, and we want to test the hypothesis a=0 (i.e. A has no effect). Let S0 be the average value of X for all observations with A=0, and S1 be the average for A=1. Because of the uncontrolled random variation E, we would not expect S0 to be equal to S1, even if A had no real effect on X. What we need to do is first estimate the variation due to E, and compare |S0-S1| to the difference we would thus expect from pure error. We can estimate the variation of E by making more than one observation of X at each combination of values of the controlled variables. These multiple observations are called <u>replicates</u>. If we assume that the error term is normally distributed, then we can use the tabled values of the F-distribution to decide whether or not a difference between S0 and S1 is large enough that it is unlikely that it is the result of pure chance. Extensions to more complicated cases are not difficult. Suppose, for example, that B is controlled to ten values (say 0,1,2,...,9). Let Ti be the average of the observations with B=i. Then we measure the variation possibly due to B by the sample variance of the Ti's, by a sum-of-squares computation. We can compare that variation to the variation from E. If the variation among the Ti's is much larger than

that predicted from pure error, then we conclude that B has a significant effect. Again we use values of the F-distribution to determine the decision criteria. The <u>significance level</u> of the decision criterion is the probability of concluding that the effect is significant, if indeed it is not. An excellent discussion of analysis of variance, along with all necessary computational formulas and tables, may be found in [23], or any other good handbook of experimetal statistics.

Confidence Intervals

In experimental statistics we often know the type of distribution from which we are sampling, but we want to determine some of its controlling parameters. For example, we often know (or assume) that we have a normal (Gaussian) distribution, but do not know its mean or variance. We then have a *two-parameter family*, and we wish to establish the parameters. For simplicity, consider a one-parameter family $f(p)$. We *sample* from f by making several observations of objects in the distribution, and *estimate* p from the observations. The mathematical form of the estimate depends on the form of the family. If f were a class of normal distributions all with variance=1, and p were the mean, then the best estimate of p would be the arithmetic mean of the observations. If f were the class of uniform distributions on the interval $[p,1]$, then a good estimate for p would be the minimum observation. In any case, once we have the estimate, we like to ask ourselves how accurate the estimate is. The question is often answered with a confidence interval. If we sample from f and estimate $p=p$, we like to also say "there is a 95% probability that $p0 \le p \le p1$", where $p0$ and $p1$ are also computed from the observations. The interpretation of this statement is important. p is not a random variable; either it is in the interval or it is not. The random variables are $p0$ and $p1$. A more accurate statement would be "experimental procedure S produced the

interval [p0,p1], and there is a 95% probability that S will produce an interval containing the true value of p".

The family of distributions underlying the coupling experiments is the binomial distribution.

$$P_p(k) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}$$

if k is an integer between 0 and n

$P_p(k)=0$ otherwise.

Here n is the sample size, k is the number of successes in the sample, and p is the probability of success on any one observation. ("Success" here can mean anything we want.) In our experiments, n is on the order of 10,000 to 50,000, and p is the fraction of all complex errors of a given type that would not be equivalent or eliminated by the test data provided, and k is the number of complex errors in the sample that are not equivalent or eliminated.

Let p0 be the value (found by iteration) such that

$$\sum_{i=0}^{k-1} P_{p0}(i) = 0.975$$

Then

$$P(p0 \leq p) = P\left(\sum_{i=0}^{k-1} P_{p0}(i) \geq \sum_{i=0}^{k-1} P_p(i)\right)$$

$$= \sum_{i=0}^{ku} P_p(i)$$

Where ku is tha largest integer such that

$$\sum_{i=0}^{ku-1} P_p(i) \leq 0.975$$

So $P(p0 \leq p) \geq 0.975$. By an analogous argument, $P(p1 \geq p) \geq 0.975$. Our 95% confidence interval is thus [p0,p1].

APPENDIX F

Program Listings

PROGRAM 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID. POQAACA.
AUTHOR. CPT R W MOREHEAD.
INSTALLATION. HQS USACSC.
DATE-WRITTEN. OCT 1973.
REMARKS.
    THIS PROGRAM PRINTS OUT A LIST OF CHANGES IN THE ETF.
    ALL ETF CHANGES WERE PROCESSED PRIOR TO THIS PROGRAM.  THE
    OLD ETF AND THE NEW ETF ARE THE INPUTS.  BUT THERE IS NO
    FURTHER PROCESSING OF THE ETF HERE.  THE ONLY OUTPUT IS A
    LISTING OF THE ADDS, CHANGES, AND DELETES.  THIS PROGRAM IS
    FOR HQ USE ONLY AND HAS NO APPLICATION IN THE FIELD.
     *****************
     MODIFIED FOR TESTING UNDER CPMS BY ALLEN ACREE
     JULY, 1979.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PRIME.
OBJECT-COMPUTER. PRIME.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OLD-ETF ASSIGN INPUT4.
    SELECT NEW-ETF ASSIGN INPUT8.
    SELECT PRNTR ASSIGN TO OUTPUT9.
DATA DIVISION.
FILE SECTION.
FD  OLD-ETF
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS OLD-REC.
01  OLD-REC.
    03  FILLER                          PIC X.
    03  OLD-KEY                         PIC X(12).
    03  FILLER                          PIC X(67).
FD  NEW-ETF
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS NEW-REC.
01  NEW-REC.
    03  FILLER                          PIC X.
    03  NEW-KEY                         PIC X(12).
    03  FILLER                          PIC X(67).
FD  PRNTR
    RECORD CONTAINS 40 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS PRNT-LINE.
01  PRNT-LINE                           PIC X(40).
WORKING-STORAGE SECTION.
01  PRNT-WORK-AREA.
    03  LINE1                           PIC X(30).
    03  LINE2                           PIC X(30).
    03  LINE3                           PIC X(20).
01  PRNT-OUT-OLD.
    03  WS-LN-1.
        05  FILLER                      PIC X VALUE SPACE.
        05  FILLER                      PIC XXXX VALUE 'O   '.
        05  LN1                         PIC X(30).
        05  FILLER                      PIC XXX VALUE SPACES.
    03  WS-LN-2.
        05  FILLER                      PIC X VALUE SPACE.
        05  FILLER                      PIC XXXX VALUE 'L   '.
```

```
               05  LN2                              PIC X(30).
               05  FILLER                           PIC XXX VALUE SPACES.
           03  WS-LN-3.
               05  FILLER                           PIC X VALUE SPACE.
               05  FILLER                           PIC XXXX VALUE 'D   '.
               05  LN3                              PIC X(20).
               05  FILLER                           PIC XXX  VALUE SPACE.
       01  PRNT-NEW-OUT.
           03  NEW-LN-1.
               05  FILLER                           PIC XXXXX VALUE ' N   '.
               05  N-LN1                            PIC X(30).
               05  FILLER                           PIC XXX VALUE SPACE.
           03  NEW-LN-2.
               05  FILLER                           PIC XXXXX VALUE ' E   '.
               05  N-LN2                            PIC X(30).
               05  FILLER                           PIC XXX VALUE SPACES.
           03  NEW-LN-3.
               05  FILLER                           PIC XXXXX VALUE ' W   '.
               05  N-LN3                            PIC X(20).
               05  FILLER                           PIC XXX VALUE SPACES.
       PROCEDURE DIVISION.
       0100-OPENS.
           OPEN INPUT OLD-ETF NEW-ETF.
           OPEN OUTPUT PRNTR.
       0110-OLD-READ.
           READ OLD-ETF AT END GO TO 0160-OLD-EOF.
       0120-NEW-READ.
           READ NEW-ETF AT END GO TO 0170-NEW-EOF.
       0130-COMPARES.
           IF OLD-KEY = NEW-KEY
               NEXT SENTENCE
           ELSE GO TO 0140-CK-ADD-DEL.
           IF OLD-REC = NEW-REC
               GO TO 0110-OLD-READ.
           MOVE OLD-REC TO PRNT-WORK-AREA.
           PERFORM 0210-OLD-WRT THRU 0210-EXIT.
           MOVE NEW-REC TO PRNT-WORK-AREA.
           PERFORM 0200-NW-WRT THRU 0200-EXIT.
           GO TO 0110-OLD-READ.
       0140-CK-ADD-DEL.
           IF OLD-KEY > NEW-KEY
               MOVE NEW-REC TO PRNT-WORK-AREA
               PERFORM 0200-NW-WRT THRU 0200-EXIT
               GO TO 0120-NEW-READ
           ELSE GO TO 0150-CK-ADD-DEL.
       0150-CK-ADD-DEL.
           MOVE OLD-REC TO PRNT-WORK-AREA.
           PERFORM 0210-OLD-WRT THRU 0210-EXIT.
           READ OLD-ETF AT END
               MOVE NEW-REC TO PRNT-WORK-AREA
               PERFORM 0200-NW-WRT THRU 0200-EXIT
               GO TO 0160-OLD-EOF.
           GO TO 0130-COMPARES.
       0160-OLD-EOF.
           READ NEW-ETF AT END GO TO 0180-EOJ.
           MOVE NEW-REC TO PRNT-WORK-AREA.
           PERFORM 0200-NW-WRT THRU 0200-EXIT.
           GO TO 0160-OLD-EOF.
       0170-NEW-EOF.
           MOVE OLD-REC TO PRNT-WORK-AREA.
           PERFORM 0210-OLD-WRT THRU 0210-EXIT.
           READ OLD-ETF AT END GO TO 0180-EOJ.
           GO TO 0170-NEW-EOF.
       0180-EOJ.
```

```
           CLOSE OLD-ETF NEW-ETF PRNTR.
           STOP RUN.
       0200-NW-WRT.
           MOVE LINE1 TO N-LN1.
           MOVE LINE2 TO N-LN2.
           MOVE LINE3 TO N-LN3.
           WRITE PRNT-LINE FROM NEW-LN-1 AFTER ADVANCING 2.
           WRITE PRNT-LINE FROM NEW-LN-2 AFTER ADVANCING 1.
           WRITE PRNT-LINE FROM NEW-LN-3 AFTER ADVANCING 1.
       0200-EXIT.
           EXIT.
       0210-OLD-WRT.
           MOVE LINE1 TO LN1.
           MOVE LINE2 TO LN2.
           MOVE LINE3 TO LN3.
           WRITE PRNT-LINE FROM WS-LN-1 AFTER ADVANCING 2.
           WRITE PRNT-LINE FROM WS-LN-2 AFTER ADVANCING 1.
           WRITE PRNT-LINE FROM WS-LN-3 AFTER ADVANCING 1.
       0210-EXIT.
           EXIT.
```

PROGRAM 2

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    PROG-1.
AUTHOR.
    JAMES L. BINGHAM.
DATE-WRITTEN.
    APRIL 14, 1979.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PRIME.
OBJECT-COMPUTER. PRIME.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT IN-TRANSACTION ASSIGN TO INPUTO.
    SELECT OUTPUT-PAYMENT ASSIGN TO OUTPUTO.

DATA DIVISION.
FILE SECTION.

FD  IN-TRANSACTION
    RECORD CONTAINS 18 CHARACTERS,
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS TRANSACTION-RECORD.
01  TRANSACTION-RECORD.
    05 ACCT-NUM                          PIC 9(8).
    05 BILLED-AMT                        PIC 9(5)V99.
    05 PERCENTAGE                        PIC V99.
    05 ACCT-CLASS                        PIC X.

FD  OUTPUT-PAYMENT
    RECORD CONTAINS 55 CHARACTERS,
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS OUTPUT-RECORD.
01  OUTPUT-RECORD                         PIC X(55).

WORKING-STORAGE SECTION.

01  W-TOTALS-OUTPUT-RECORD.
    05 FILLER                            PIC X(4) VALUE SPACES.
    05 NAME-OF-CLASS                     PIC X(34).
    05 TOTAL-CLASS-PAY                   PIC $$$$$$9.99.
    05 FILLER                            PIC X(4) VALUE SPACES.

01  W-OUTPUT-RECORD.
    05 FILLER                            PIC XXX VALUE SPACES.
    05 W-ACCT-NUM                        PIC 9(8).
    05 FILLER                            PIC XXX VALUE SPACES.
    05 W-BILLED-AMT                      PIC 9(5).99.
    05 FILLER                            PIC XXX VALUE SPACES.
    05 W-PERCENTAGE                      PIC .99.
    05 FILLER                            PIC XXX VALUE SPACES.
    05 W-ACCT-CLASS                      PIC X.
    05 FILLER                            PIC XXX VALUE SPACES.
    05 W-PAYMENT                         PIC $$$$$9.99.

01  TEMPORARY-ITEMS.
    05 TOTAL-A-PAY                       PIC 9(6)V99.
    05 TOTAL-X-PAY                       PIC 9(6)V99.
    05 TOTAL-M-PAY                       PIC 9(6)V99.
    05 TOTAL-T-PAY                       PIC 9(6)V99.
```

```
           05 TOTAL-Z-PAY                        PIC 9(6)V99.
           05 PAY-AMT-A                          PIC 9(5)V99.
           05 PAY-AMT-X                          PIC 9(5)V99.
           05 PAY-AMT-M                          PIC 9(5)V99.
           05 PAY-AMT-T                          PIC 9(5)V99.
           05 PAY-AMT-Z                          PIC 9(5)V99.

       01  ERROR-MESSAGE.
           05 INVALID-DATA-RECORD                 PIC X(50)
             VALUE 'INVALID DATA ON THIS CARD'.

       01  FLAG-VALUE.
           05 MORE-DATA-REMAINS                   PIC X VALUE 'Y'.
             88 NO-MORE-DATA-REMAINS                   VALUE 'N'.

       PROCEDURE DIVISION.
       PROCESS-TRANSACTION.
           OPEN INPUT IN-TRANSACTION
                OUTPUT OUTPUT-PAYMENT.
           MOVE ZEROES TO TOTAL-A-PAY, TOTAL-X-PAY, TOTAL-M-PAY,
                          TOTAL-T-PAY, TOTAL-Z-PAY.
           READ IN-TRANSACTION
                AT END MOVE 'N' TO MORE-DATA-REMAINS.
           PERFORM CHECK-DATA UNTIL MORE-DATA-REMAINS = 'N'.
           PERFORM WRITE-OUTPUT-TOTALS.
           CLOSE IN-TRANSACTION
                 OUTPUT-PAYMENT.
           STOP RUN.

       CHECK-DATA.
           IF     ACCT-NUM    IS NUMERIC
              AND BILLED-AMT  IS NUMERIC
              AND PERCENTAGE  IS NUMERIC
              AND  (ACCT-CLASS = 'A' OR
                    ACCT-CLASS = 'X' OR
                    ACCT-CLASS = 'M' OR
                    ACCT-CLASS = 'T' OR
                    ACCT-CLASS = 'Z')
              PERFORM PROCESS-ONE-TRANSACTION
           ELSE
              WRITE OUTPUT-RECORD FROM ERROR-MESSAGE.
           READ IN-TRANSACTION
                AT END MOVE 'N' TO MORE-DATA-REMAINS.

       PROCESS-ONE-TRANSACTION.
           MOVE ACCT-NUM   TO W-ACCT-NUM.
           MOVE BILLED-AMT TO W-BILLED-AMT.
           MOVE PERCENTAGE TO W-PERCENTAGE.
           MOVE ACCT-CLASS TO W-ACCT-CLASS.

           IF ACCT-CLASS = 'A' OR ACCT-CLASS = 'X'
               COMPUTE PERCENTAGE = 1.00 - PERCENTAGE
               IF ACCT-CLASS = 'A'
                   MULTIPLY BILLED-AMT BY PERCENTAGE
                            GIVING PAY-AMT-A ROUNDED
                   ADD PAY-AMT-A TO TOTAL-A-PAY
                   MOVE PAY-AMT-A TO W-PAYMENT
               ELSE
                   MULTIPLY BILLED-AMT BY PERCENTAGE
                            GIVING PAY-AMT-X ROUNDED
                   ADD PAY-AMT-X TO TOTAL-X-PAY
                   MOVE PAY-AMT-X TO W-PAYMENT.

           IF ACCT-CLASS = 'M'
```

```
        MULTIPLY BILLED-AMT BY PERCENTAGE
                 GIVING PAY-AMT-M ROUNDED
        ADD PAY-AMT-M TO TOTAL-M-PAY
        MOVE PAY-AMT-M TO W-PAYMENT.

    IF ACCT-CLASS = 'T'
        MOVE BILLED-AMT TO PAY-AMT-T
        ADD PAY-AMT-T TO TOTAL-T-PAY
        MOVE PAY-AMT-T TO W-PAYMENT.

    IF ACCT-CLASS = 'Z'
        MOVE BILLED-AMT TO PAY-AMT-Z
        ADD PAY-AMT-Z TO TOTAL-Z-PAY
        MOVE PAY-AMT-Z TO W-PAYMENT.

    WRITE OUTPUT-RECORD FROM W-OUTPUT-RECORD.

WRITE-OUTPUT-TOTALS.
    MOVE TOTAL-A-PAY TO TOTAL-CLASS-PAY.
    MOVE '  TOTAL AMOUNT FOR CLASS A:  ' TO NAME-OF-CLASS.
    WRITE OUTPUT-RECORD FROM W-TOTALS-OUTPUT-RECORD.

    MOVE TOTAL-X-PAY TO TOTAL-CLASS-PAY.
    MOVE '  TOTAL AMOUNT FOR CLASS X:  ' TO NAME-OF-CLASS.
    WRITE OUTPUT-RECORD FROM W-TOTALS-OUTPUT-RECORD.

    MOVE TOTAL-M-PAY TO TOTAL-CLASS-PAY.
    MOVE '  TOTAL AMOUNT FOR CLASS M:  ' TO NAME-OF-CLASS.
    WRITE OUTPUT-RECORD FROM W-TOTALS-OUTPUT-RECORD.

    MOVE TOTAL-T-PAY TO TOTAL-CLASS-PAY.
    MOVE '  TOTAL AMOUNT FOR CLASS T:  ' TO NAME-OF-CLASS.
    WRITE OUTPUT-RECORD FROM W-TOTALS-OUTPUT-RECORD.

    MOVE TOTAL-Z-PAY TO TOTAL-CLASS-PAY.
    MOVE '  TOTAL AMOUNT FOR CLASS Z:  ' TO NAME-OF-CLASS.
    WRITE OUTPUT-RECORD FROM W-TOTALS-OUTPUT-RECORD.
```

PROGRAM 3

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE-4.
REMARKS.  ADAPTED FROM YOURDAN, ET AL. "LEARNING TO PROGRAM
          IN STRUCTURED COBOL."
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  PRIME.
OBJECT-COMPUTER.  PRIME.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT APPLICATION-CARDS-FILE ASSIGN TO INPUTO.
    SELECT PROFILE-LISTING        ASSIGN TO OUTPUTO.

DATA DIVISION.
FILE SECTION.

FD  APPLICATION-CARDS-FILE
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS NAME-ADDRESS-AND-PHONE-IN.
01  NAME-ADDRESS-AND-PHONE-IN.
    05  NAME-IN                           PIC X(20).
    05  ADDRESS-IN                        PIC X(40).
    05  PHONE-IN                          PIC X(11).
    05  FILLER                            PIC X(3).
    05  ACCT-NUM-IN1                      PIC 9(6).

FD  PROFILE-LISTING
    RECORD CONTAINS 132 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS PRINT-LINE-OUT.
01  PRINT-LINE-OUT                        PIC X(132).

WORKING-STORAGE SECTION.
01  COMMON-WS.
    05  CARDS-LEFT                        PIC X(3).
01  CREDIT-INFORMATION-IN.
    05  CARD-TYPE-IN                      PIC X.
    05  ACCT-NUM-IN2                      PIC 9(6).
    05  FILLER                            PIC X.
    05  CREDIT-INFO-IN                    PIC X(22).
    05  FILLER                            PIC X(50).
01  APPLICATION-DATA-WSB1.
    05  NAME-AND-ADDRESS-WS.
        10  NAME-WS                       PIC X(20).
        10  ADDRESS-WS.
            15  STREET-WS                 PIC X(20).
            15  CITY-WS                   PIC X(13).
            15  STATE-WS                  PIC XX.
            15  ZIP-WS                    PIC X(5).
    05  PHONE-WS.
        10  AREA-CODE-WS                  PIC 9(3).
        10  NUMBR-WS                      PIC X(8).
    05  FILLER                            PIC X(3).
    05  ACCT-NUM-WS                       PIC 9(6).
    05  CREDIT-INFO-WS.
        10  SEX-WS                        PIC X.
        10  FILLER                        PIC X.
        10  MARITAL-STATUS-WS             PIC X.
        10  FILLER                        PIC X.
        10  NUMBER-DEPENS-WS              PIC X.
```

```
               10  FILLER                           PIC X.
               10  INCOME-HUNDREDS-WS               PIC 9(3).
               10  FILLER                           PIC X.
               10  YEARS-EMPLOYED-WS                PIC 99.
               10  FILLER                           PIC X.
               10  OWN-OR-RENT-WS                   PIC X.
               10  FILLER                           PIC X.
               10  MORTGAGE-OR-RENTAL-WS            PIC 9(3).
               10  FILLER                           PIC X.
               10  OTHER-PAYMENTS-WS                PIC 9(3).
       01  DISCR-INCOME-CALC-FIELDS-WSC8.
           05  ANNUAL-INCOME-WS                     PIC 9(5).
           05  ANNUAL-TAX-WS                        PIC 9(5).
           05  TAX-RATE-WS                          PIC 9V99   VALUE 0.25.
           05  MONTHS-IN-YEAR                       PIC 99     VALUE 12.
           05  MONTHLY-NET-INCOME-WS                PIC 9(4).
           05  MONTHLY-PAYMENTS-WS                  PIC 9(4).
           05  DISCR-INCOME-WS                      PIC S9(3).

       01  LINE-1-WSB3.
           05  FILLER                               PIC X(5) VALUE SPACES.
           05  NAME-L1                              PIC X(20).
           05  FILLER                               PIC X(11)
                   VALUE '    PHONE ('.
           05  AREA-CODE-L1                         PIC 9(3).
           05  FILLER                               PIC XX  VALUE ') '.
           05  NUMBR-L1                             PIC X(8).
           05  FILLER                               PIC X(3) VALUE SPACES.
           05  SEX-L1                               PIC X(6).
           05  FILLER                               PIC X(9) VALUE SPACES.
           05  FILLER                               PIC X(14)
                    VALUE 'INCOME       $'.
           05  INCOME-HUNDREDS-L1                   PIC 9(3).
           05  FILLER                               PIC X(28)
                    VALUE '00 PER YEAR; IN THIS EMPLOY '.
           05  YEARS-EMPLOYED-L1.
               10  YEARS-L1                         PIC XX.
               10  DESCN-L1                         PIC X(16).
       01  LINE-2-WSB3.
           05  FILLER                               PIC X(5) VALUE SPACES.
           05  STREET-L2                            PIC X(20).
           05  FILLER                               PIC X(27) VALUE SPACES.
           05  MARITAL-STATUS-L2                    PIC X(8).
           05  FILLER                               PIC X(7) VALUE SPACES.
           05  OUTGO-DESCN                          PIC X(16).
           05  MORTGAGE-OR-RENTAL-L2                PIC 9(3).
           05  FILLER                               PIC X(11)
                 VALUE ' PER MTH   '.
           05  FILLER                               PIC X(22)
                 VALUE 'DISCRETIONARY INCOME $'.
           05  DISCR-INCOME-L2                      PIC 9(3).
           05  FILLER                               PIC X(9)
                 VALUE ' PER MTH '.
       01  LINE-3-WSB3.
           05  FILLER                               PIC X(5) VALUE SPACES.
           05  CITY-L3                              PIC X(13).
           05  FILLER                               PIC X VALUE SPACE.
           05  STATE-L3                             PIC XX.
           05  FILLER                               PIC X VALUE SPACE.
           05  ZIP-L3                               PIC X(5).
           05  FILLER                               PIC X(7) VALUE '  A/C: '.
           05  ACCT-NUM-L3                          PIC 9(6).
           05  FILLER                               PIC X(12) VALUE SPACES.
           05  NUMBER-DEPENS-L3                     PIC 9.
```

```
           05  FILLER                                PIC X(14)
                   VALUE ' DEPENDENTS    '.
           05  FILLER                                PIC X(16)
                   VALUE 'OTHER PAYMENTS $'.
           05  OTHER-PAYMENTS-L3                     PIC 9(3).

       PROCEDURE DIVISION.
       A0-MAIN-BODY.
           PERFORM A1-INITIALIZATION.
           PERFORM A2-PRINT-PROFILES
             UNTIL CARDS-LEFT = 'NO '.
           PERFORM A3-END-OF-JOB.
           STOP RUN.

       A1-INITIALIZATION.
           OPEN INPUT     APPLICATION-CARDS-FILE
                OUTPUT    PROFILE-LISTING.
      *** USELESS INITIALIZATIONS HAVE BEEN COMMENTED OUT
      ***  MOVE ZEROES TO ANNUAL-INCOME-WS.
      ***  MOVE ZEROES TO ANNUAL-TAX-WS.
      ***  MOVE ZEROES TO MONTHLY-NET-INCOME-WS.
      ***  MOVE ZEROES TO MONTHLY-PAYMENTS-WS.
      ***  MOVE ZEROES TO DISCR-INCOME-WS.
           MOVE 'YES' TO CARDS-LEFT.
           READ APPLICATION-CARDS-FILE
               AT END MOVE 'NO ' TO CARDS-LEFT.
      *  THE FIRST CARD OF A PAIR IS NOW IN THE BUFFER.

       A2-PRINT-PROFILES.
           PERFORM B1-GET-A-PAIR-OF-CARDS-INTO-WS.
           PERFORM B2-CALC-DISCRETNRY-INCOME.
           PERFORM B3-ASSEMBLE-PRINT-LINES.
           PERFORM B4-WRITE-PROFILE.

       A3-END-OF-JOB.
           CLOSE APPLICATION-CARDS-FILE
                 PROFILE-LISTING.

       B1-GET-A-PAIR-OF-CARDS-INTO-WS.
           MOVE NAME-IN TO NAME-WS.
           MOVE ADDRESS-IN TO ADDRESS-WS.
           MOVE PHONE-IN TO PHONE-WS.
           MOVE ACCT-NUM-IN1 TO ACCT-NUM-WS.
           READ APPLICATION-CARDS-FILE INTO CREDIT-INFORMATION-IN
      ***      AT END MOVE 'NO ' TO CARDS-LEFT.
               AT END MOVE '    *** MISSING SECOND CARD OF PAIR ***'
                           TO PRINT-LINE-OUT
                      WRITE PRINT-LINE-OUT AFTER ADVANCING 2 LINES
                      PERFORM A3-END-OF-JOB
                      STOP RUN.
      * THE SECOND CARD OF THE PAIR IS NOW IN THE BUFFER.
           MOVE CREDIT-INFO-IN TO CREDIT-INFO-WS
           READ APPLICATION-CARDS-FILE
               AT END MOVE 'NO ' TO CARDS-LEFT.
      * THE FIRST CARD OF THE NEXT PAIR IS NOW IN THE BUFFER.

       B2-CALC-DISCRETNRY-INCOME.
           COMPUTE ANNUAL-INCOME-WS = INCOME-HUNDREDS-WS * 100.
           COMPUTE ANNUAL-TAX-WS    = ANNUAL-INCOME-WS * TAX-RATE-WS.
           COMPUTE MONTHLY-NET-INCOME-WS ROUNDED
               = (ANNUAL-INCOME-WS - ANNUAL-TAX-WS) / MONTHS-IN-YEAR.
           COMPUTE MONTHLY-PAYMENTS-WS = MORTGAGE-OR-RENTAL-WS
                               + OTHER-PAYMENTS-WS.
           COMPUTE DISCR-INCOME-WS = MONTHLY-NET-INCOME-WS
```

```
                                - MONTHLY-PAYMENTS-WS
           ON SIZE ERROR MOVE 999 TO DISCR-INCOME-WS.
*     DISCRETIONARY INCOMES OVER $999 PER MONTH ARE SET AT $999.

 B3-ASSEMBLE-PRINT-LINES.
       MOVE NAME-WS TO NAME-L1.
       MOVE STREET-WS TO STREET-L2.
       MOVE CITY-WS TO CITY-L3.
       MOVE STATE-WS TO STATE-L3.
       MOVE ZIP-WS TO ZIP-L3.
       MOVE AREA-CODE-WS TO AREA-CODE-L1.
       MOVE NUMBR-WS TO NUMBR-L1.
       MOVE ACCT-NUM-WS TO ACCT-NUM-L3.
       IF SEX-WS = 'M' MOVE 'MALE  ' TO SEX-L1.
       IF SEX-WS = 'F' MOVE 'FEMALE' TO SEX-L1.
       IF MARITAL-STATUS-WS = 'S' MOVE 'SINGLE  '
               TO MARITAL-STATUS-L2.
       IF MARITAL-STATUS-WS = 'M' MOVE 'MARRIED '
               TO MARITAL-STATUS-L2.
       IF MARITAL-STATUS-WS = 'D' MOVE 'DIVORCED'
               TO MARITAL-STATUS-L2.
       IF MARITAL-STATUS-WS = 'W' MOVE 'WIDOWED '
               TO MARITAL-STATUS-L2.
       MOVE NUMBER-DEPENS-WS TO NUMBER-DEPENS-L3.
       MOVE INCOME-HUNDREDS-WS TO INCOME-HUNDREDS-L1.
       IF  YEARS-EMPLOYED-WS IS EQUAL TO 0
           MOVE 'LESS THAN 1 YEAR' TO YEARS-EMPLOYED-L1
       ELSE
           MOVE YEARS-EMPLOYED-WS TO YEARS-L1
           MOVE ' YEARS      ' TO DESCN-L1.
       IF OWN-OR-RENT-WS = 'O' MOVE 'MORTGAGE:      $'
               TO OUTGO-DESCN.
       IF OWN-OR-RENT-WS = 'R' MOVE 'RENTAL:        $'
               TO OUTGO-DESCN.
       MOVE MORTGAGE-OR-RENTAL-WS TO MORTGAGE-OR-RENTAL-L2.
       MOVE OTHER-PAYMENTS-WS TO OTHER-PAYMENTS-L3.
       MOVE DISCR-INCOME-WS TO DISCR-INCOME-L2.

 B4-WRITE-PROFILE.
***    MOVE SPACES TO PRINT-LINE-OUT.
       WRITE PRINT-LINE-OUT FROM LINE-1-WSB3
               AFTER ADVANCING 4 LINES.
***    MOVE SPACES TO PRINT-LINE-OUT.
       WRITE PRINT-LINE-OUT FROM LINE-2-WSB3
               AFTER ADVANCING 1 LINES.
***    MOVE SPACES TO PRINT-LINE-OUT.
       WRITE PRINT-LINE-OUT FROM LINE-3-WSB3
               AFTER ADVANCING 1 LINES.
```

PROGRAM 4

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SRMFREP.
AUTHOR. R A OVERBEEK.
REMARKS. THIS PROGRAM IS USED TO PRODUCE THE STATUS REPORTS
         BY DEPARTMENT, FOR ALL OF THE STUDENTS RECORDED IN
         THE SRMF.

         ADAPTED TO THE COBCL MUTATION SYSTEM BY ALLEN ACREE.

               ERRORS DISCOVERED:

               (1)  ERRORS IN THE INPUT FILE SETUP, CHECKED FOR
               IN THE PROGRAM, CAUSE REFERENCES TO UNDEFINED
               DATA, PARTICULARLY LINE-COUNT.  CORRECTED WITH
               A VALUE CLAUSE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. CMS.
OBJECT-COMPUTER. CMS.
SPECIAL-NAMES. C01 IS TOP-OF-PAGE.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER ASSIGN TO INPUTO.
    SELECT PRINT-FILE ASSIGN TO OUTPUTO.

DATA DIVISION.
FILE SECTION.
FD  MASTER
    RECORD CONTAINS 141 CHARACTERS,
    LABEL RECORDS ARE STANDARD,
    DATA RECORD IS ITEM.
01  ITEM.
    02  SOC-SEC-IN.
        03  SOC-SEC-IN-1              PIC X(3).
        03  SOC-SEC-IN-2              PIC X(2).
        03  SOC-SEC-IN-3              PIC X(4).
    02  NAME-IN                       PIC X(5).
    02  ADDR-IN-1                     PIC X(5).
    02  ADDR-IN-2                     PIC X(5).
    02  MAJOR-IN                      PIC X(4).
    02  STATUS-IN                     PIC X(1).
    02  NO-COURSES                    PIC 99.
    02  COURSE-ENTRY OCCURS 11 TIMES.
        03  DEPT-OFF                  PIC X(2).
        03  COURSE-NO                 PIC X(2).
        03  CREDITS                   PIC 99.
        03  SEMESTER                  PIC X(1).
        03  YEAR                      PIC X(2).
        03  GRADE                     PIC X(1).
FD  PRINT-FILE
    RECORD CONTAINS 89 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS PRINT-BUFF.
01  PRINT-BUFF                        PIC X(89).

WORKING-STORAGE SECTION.
77  END-ALL                           PIC 99.
77  END-MARKER                        PIC 99.
77  P-INDEX                           PIC 9.
77  POINTS                            PIC 999.
77  CR-HRS                            PIC 999.
```

```
       77  INCR                              PIC 99.
       77  C-INDEX                           PIC 99.
       77  PAGE-NO                           PIC 999 VALUE IS 1.
       77  LINE-COUNT                        PIC 99  VALUE ZERO.
       77  SAVE-KEY                          PIC X(4).
       77  TOT-NO-RECORDS                    PIC 9999999 VALUE IS 0.
       77  SUB-TOT-NO                        PIC 9999999.

       01  HEADER.
           02  FILLER                        PIC X(14).
           02  COLLEGE                       PIC X(30).
           02  DATE-IN                       PIC X(8).
       01  TRAILER.
           02  FILLER                        PIC X(49).
           02  NO-RECORDS                    PIC 9999999.
       01  PRINT-LINE.
           02  FILLER                        PIC X(1).
           02  SOC-SEC-OUT.
               03  SOC-SEC-01                PIC X(3).
               03  SOC-SEC-F1                PIC X(1).
               03  SOC-SEC-02                PIC X(2).
               03  SOC-SEC-F2                PIC X(1).
               03  SOC-SEC-03                PIC X(4).
           02  FILLER                        PIC X(2).
           02  NAME-ADDR                     PIC X(5).
           02  FILLER                        PIC X(1).
           02  MAJOR-O                       PIC X(4).
           02  FILLER                        PIC X(1).
           02  STATUS-O                      PIC X(1).
           02  FILLER                        PIC X(1).
           02  GPA                           PIC 9.99.
           02  FILLER                        PIC X(2).
           02  COURSE-O    OCCURS 3 TIMES.
               03  C-DEPT                    PIC X(2).
               03  FILLER                    PIC X(1).
               03  C-NO                      PIC X(2).
               03  FILLER                    PIC X(1).
               03  CREDITS-O                 PIC Z9.
               03  FILLER                    PIC X(1).
               03  SEMESTER-O                PIC X(1).
               03  DASH-O                    PIC X(1).
               03  YEAR-O                    PIC X(2).
               03  FILLER                    PIC X(2).
               03  GRADE-O                   PIC X(1).
               03  FILLER                    PIC X(2).
           02  FILLER                        PIC X(2).
       01  PAGE-HEADER.
           02  FILLER                        PIC X(4)  VALUE SPACES.
           02  DATE-O                        PIC X(8).
           02  FILLER                        PIC X(17)  VALUE SPACES.
           02  COLL-O                        PIC X(30).
           02  FILLER                        PIC X(17)  VALUE SPACES.
           02  FILLER                        PIC X(5)  VALUE IS 'PAGE'.
           02  PAGE-O                        PIC ZZ9.
           02  FILLER                        PIC X(5)  VALUE SPACES.
       01  COL-HDR-1.
           02  FILLER                        PIC X(20)
                   VALUE ' SOC SEC      N & A '.
           02  FILLER                        PIC X(10) VALUE 'MAJ ST GPA'.
           02  FILLER                        PIC X(9) VALUE SPACES.
           02  FILLER                        PIC X(6) VALUE 'COURSE'.
           02  FILLER                        PIC X(12) VALUE SPACES.
           02  FILLER                        PIC X(6) VALUE 'COURSE'.
           02  FILLER                        PIC X(12) VALUE SPACES.
```

```
         02  FILLER                         PIC X(6) VALUE 'COURSE'.
         02  FILLER                         PIC X(8) VALUE SPACES.
     01  COL-HDR-2.
         02  FILLER                         PIC X(33) VALUE SPACES.
         02  FILLER                         PIC X(18)
                VALUE ' NMBR CR S-YR  GR '.
         02  FILLER                         PIC X(18)
                VALUE ' NMBR CR S-YR  GR '.
         02  FILLER                         PIC X(20)
                VALUE ' NMBR CR S-YR  GR   '.
     01  SUB-TOT-LINE.
         02  FILLER                         PIC X(4)  VALUE SPACES.
         02  FILLER                         PIC X(8)
                VALUE IS 'TOTAL = '.
         02  SUB-TOT                        PIC ZZZZZZ9.
         02  FILLER                         PIC X(70)  VALUE SPACES.
     PROCEDURE DIVISION.
    * MAIN-PROGRAM SECTION.
     START.
         OPEN INPUT MASTER OUTPUT PRINT-FILE.
         READ MASTER INTO HEADER AT END GO TO EOF.
         IF SOC-SEC-IN IS = SPACES GO TO GOT-HEADER.
         MOVE ' NO HEADER FOUND ON THE MASTER FILE ***' TO PRINT-LINE.
         PERFORM PRINT2-ROUTINE THRU PRINT2-EXIT.
         GO TO CLOSE-FILES.
     GOT-HEADER.
         MOVE COLLEGE TO COLL-O.
         MOVE DATE-IN TO DATE-O.
         READ MASTER AT END GO TO EOF.
         IF SOC-SEC-IN IS NOT = '999999999' GO TO SAVE-DEPT-NAME.
         MOVE ' NO ITEM RECORDS IN MASTER FILE ***' TO PRINT-LINE.
         PERFORM PRINT2-ROUTINE THRU PRINT2-EXIT.
         GO TO CLOSE-FILES.
     SAVE-DEPT-NAME.
         MOVE MAJOR-IN TO SAVE-KEY.
    * NAME OF DEPARTMENT IS SUBTOTAL KEY.  BREAK OCCURS WHENEVER
    * FIELD IS DIFFERENT ON TWO CONSECUTIVE RECORDS.
         MOVE 0 TO SUB-TOT-NO.
         MOVE 1 TO PAGE-NO.
    * PAGE-NO IS RESET TO 1 FOR EACH DEPARTMENT REPORT.
         MOVE 16 TO LINE-COUNT.
         MOVE SPACES TO PRINT-LINE.

     ITEM-LOOP.
         PERFORM ITEM-ROUTINE THRU ITEM-EXIT.
         ADD 1 TO SUB-TOT-NO.
         READ MASTER INTO TRAILER AT END GO TO EOF.
         IF MAJOR-IN IS = SAVE-KEY GO TO ITEM-LOOP.

     DO-SUB-TOTALS.
         MOVE SUB-TOT-NO TO SUB-TOT.
         WRITE PRINT-BUFF FROM SUB-TOT-LINE AFTER ADVANCING 2 LINES.
         ADD SUB-TOT-NO TO TOT-NO-RECORDS.
         IF SOC-SEC-IN IS NOT = '999999999' GO TO SAVE-DEPT-NAME.
         MOVE TOT-NO-RECORDS TO SUB-TOT.
         WRITE PRINT-BUFF FROM SUB-TOT-LINE
                 AFTER ADVANCING TOP-OF-PAGE.
         IF NO-RECORDS IS = TOT-NO-RECORDS GO TO CLOSE-FILES.
         MOVE ' *** MASTER TRAILER VERIFICATION HAS FAILED ***'
                 TO PRINT-LINE.
         PERFORM PRINT2-ROUTINE THRU PRINT2-EXIT.
     CLOSE-FILES.
         CLOSE MASTER PRINT-FILE.
         STOP RUN.
```

```
EOF.
    MOVE ' EOF ON MASTER FILE ****' TO PRINT-LINE.
    PERFORM PRINT2-ROUTINE THRU PRINT2-EXIT.
    GO TO CLOSE-FILES.

* SUB-ROUTINE SECTION.

PRINT1-ROUTINE.
    IF LINE-COUNT IS < 16 GO TO NORMAL-PRINT.
    PERFORM HEADER-ROUTINE THRU HEADER-EXIT.
    WRITE PRINT-BUFF FROM PRINT-LINE AFTER ADVANCING 2 LINES.
    ADD 2 TO LINE-COUNT.
    GO TO COMMON-POINT.
NORMAL-PRINT.
    WRITE PRINT-BUFF FROM PRINT-LINE AFTER ADVANCING 1 LINES.
    ADD 1 TO LINE-COUNT.
COMMON-POINT.
    MOVE SPACES TO PRINT-LINE.
PRINT1-EXIT.  EXIT.

PRINT2-ROUTINE.
    IF LINE-COUNT IS > 14
        PERFORM HEADER-ROUTINE THRU HEADER-EXIT.
    WRITE PRINT-BUFF FROM PRINT-LINE AFTER ADVANCING 2 LINES.
    ADD 2 TO LINE-COUNT.
    MOVE SPACES TO PRINT-LINE.
PRINT2-EXIT.  EXIT.

HEADER-ROUTINE.
    MOVE PAGE-NO TO PAGE-O.
    WRITE PRINT-BUFF FROM PAGE-HEADER
        AFTER ADVANCING TOP-OF-PAGE.
    ADD 1 TO PAGE-NO.
    WRITE PRINT-BUFF FROM COL-HDR-1 AFTER ADVANCING 2 LINES.
    WRITE PRINT-BUFF FROM COL-HDR-2 AFTER ADVANCING 1 LINES.
    MOVE 0 TO LINE-COUNT.
HEADER-EXIT.  EXIT.

ITEM-ROUTINE.
    MOVE SOC-SEC-IN-1 TO SOC-SEC-O1.
    MOVE SOC-SEC-IN-2 TO SOC-SEC-O2.
    MOVE SOC-SEC-IN-3 TO SOC-SEC-O3.
    MOVE '-' TO SOC-SEC-F1.
    MOVE '-' TO SOC-SEC-F2.
    MOVE NAME-IN TO NAME-ADDR.
    MOVE MAJOR-IN TO MAJOR-O.
    MOVE STATUS-IN TO STATUS-O
*  CALCULATE THE GPA.
    MOVE 0 TO POINTS.
    MOVE 0 TO CR-HRS.
    PERFORM GPA-ACCUM THRU GPA-EXIT VARYING C-INDEX
        FROM 1 BY 1 UNTIL C-INDEX IS > NO-COURSES.
    IF CR-HRS IS = 0 GO TO NO-GPA.
    DIVIDE POINTS BY CR-HRS GIVING GPA ROUNDED.
*       IN THE FOLLOWING THESE INDICES ARE USED:
*       END-ALL: THE INDEX OF THE FIRST UNUSED COURSE
*                ENTRY;  THIS MARKS THE END OF THE COURSES
*                TO PRINT;
*       END-MARKER: WHEN FILL-LINE IS CALLED END-MARKER
*                POINTS AT THE FIRST COURSE ENTRY PAST THE
*                LAST ENTRY TO BE PUT INTO THE LINE;
*       C-INDEX: WHEN FILL-LINE IS CALLED C-INDEX POINTS
*                AT THE FIRST COURSE ENTRY WHICH GETS
*                PUT INTO THE PRINT-LINE;  THUS, IF C-INDEX
```

```
*                   IS EQUAL TO END-MARKER, NO COURSE ENTRIES
*                   GET PUT INTO THE PRINT LINE;
*         P-INDEX:  INDEXES THE SPOT IN THE PRINT-LINE
*                   WHERE THE ENTRY POINTED TO BY C-INDEX
*                   IS TO BE MOVED;  THUS, ITS RANGE IS 1 TO 3.

 NO-GPA.
     MOVE 1 TO C-INDEX.
     ADD 1 NO-COURSES GIVING END-ALL.
     MOVE 4 TO END-MARKER.
     IF END-ALL IS < END-MARKER MOVE END-ALL TO END-MARKER.
     PERFORM FILL-LINE THRU FILL-EXIT.
     PERFORM PRINT2-ROUTINE THRU PRINT2-EXIT.
     MOVE ADDR-IN-1 TO NAME-ADDR.
     MOVE 7 TO END-MARKER.
     IF END-ALL IS < END-MARKER MOVE END-ALL TO END-MARKER.
     PERFORM FILL-LINE THRU FILL-EXIT.
     PERFORM PRINT1-ROUTINE THRU PRINT1-EXIT.
     MOVE ADDR-IN-2 TO NAME-ADDR.
     MOVE 10 TO END-MARKER.
 COURSE-LOOP.
     IF END-ALL IS < END-MARKER MOVE END-ALL TO END-MARKER.
     PERFORM FILL-LINE THRU FILL-EXIT.
     PERFORM PRINT1-ROUTINE THRU PRINT1-EXIT.
     IF C-INDEX = END-ALL GO TO ITEM-EXIT.
     ADD 3 C-INDEX GIVING END-MARKER.
     GO TO COURSE-LOOP.
 ITEM-EXIT.  EXIT.
 FILL-LINE.
     MOVE 1 TO P-INDEX.
 CHECK-END.
     IF C-INDEX IS = END-MARKER GO TO FILL-EXIT.
     MOVE DEPT-OFF (C-INDEX) TO C-DEPT (P-INDEX).
     MOVE COURSE-NO (C-INDEX) TO C-NO (P-INDEX).
     MOVE CREDITS (C-INDEX) TO CREDITS-O (P-INDEX).
     MOVE SEMESTER (C-INDEX) TO SEMESTER-O (P-INDEX).
     MOVE '-' TO DASH-O (P-INDEX).
     MOVE YEAR (C-INDEX) TO YEAR-O (P-INDEX).
     MOVE GRADE (C-INDEX) TO GRADE-O (P-INDEX).
     ADD 1 TO C-INDEX.
     ADD 1 TO P-INDEX.
     GO TO CHECK-END.
 FILL-EXIT.  EXIT.

 GPA-ACCUM.
     IF GRADE (C-INDEX) IS NOT = 'A' GO TO NOTA.
     MULTIPLY CREDITS (C-INDEX) BY 4 GIVING INCR.
     GO TO COMMON-ADD.
 NOTA.
     IF GRADE (C-INDEX) IS NOT = 'B' GO TO NOTB.
     MULTIPLY CREDITS (C-INDEX) BY 3 GIVING INCR.
     GO TO COMMON-ADD.
 NOTB.
     IF GRADE (C-INDEX) IS NOT = 'C' GO TO NOTC.
     MULTIPLY CREDITS (C-INDEX) BY 2 GIVING INCR.
     GO TO COMMON-ADD.
 NOTC.
     IF GRADE (C-INDEX) IS NOT = 'D' GO TO NOTD.
     MULTIPLY CREDITS (C-INDEX) BY 1 GIVING INCR.
     GO TO COMMON-ADD.
 NOTD.
     IF GRADE (C-INDEX) IS NOT = 'F' GO TO GPA-EXIT.
     MOVE 0 TO INCR.
 COMMON-ADD.
```

```
     ADD INCR TO POINTS.
     ADD CREDITS (C-INDEX) TO CR-HRS.
GPA-EXIT.  EXIT.
```

PROGRAM 5

```
IDENTIFICATION DIVISION.
*
*    REPORT CONTAINS THE INPUT DATA ALONG WITH THE
*    CURRENT COMMISSION FOR EACH SALESMAN. AT THE
*    END OF THIS SINGLE SPACED REPORT THE FOLLOWING
*    TOTALS ARE PRINTED: YEAR TO DATE SALES, CUR-
*    RENT SALES, CURRENT COMMISSION.
*
*    CURRENT COMMISSION IS CALCULATED AS FOLLOWS:
*      CURRENT-COMMISSION = CURRENT-SALES *
*        ( COMMISSION-RATE + VOLUME-BONUS + DEPARTMENT-BONUS )
*
*    WITH DEPARTMENT BONUS DETERMINED AS FOLLOWS:
*        DEPT      BONUS
*         01        0.1%
*         02        0.1%
*         04        0.7%
*         05        0.6%
*         06        0.4%
*         07        0.6%
*         09        0.4%
*        OTHER      0.0%
*
*    WITH VOLUME BONUS DETERMINED AS FOLLOWS:
*        AVERAGE MONTHLY SALES      BONUS
*          UNDER $500                0.0%
*          $500 TO $999.99           0.3%
*          $1000 TO $1999.99         0.4%
*          OVER $2000                0.6%
*
*    WITH AVERAGE MONTHS SALES DETERMINED AS FOLLOWS:
*      AVERAGE-MONTHLY-SALES =
*        ( YEAR-TO-DATE-SALES + CURRENT-SALES ) / MONTHS-EMPLOYED

PROGRAM-ID.  COMMISSION-REPORT.

AUTHOR.
    DANIEL CASTAGNO,ICS 3400,STUDENT NUMBER 654,PROGRAM 1.

REMARKS.  SLIGHTLY MODIFIED FOR CMS.1 BY A.ACREE.
          MUTATION TESTING UNCOVERED THE FOLLOWING ERRORS AND
          INEFFICIENCIES:
          (1) REPORT HEADER WITH PAGE ADVANCE WAS NOT PRINTED
          AFTER FULL-PAGE CONDITION RAISED BY INVALID DATA RECORD
          EXTRA PERFORM INSERTED.
          (2) DATA ITEMS DEFINED AND NEVER USED -- DELETED.
          (3) MOVE STATEMENT REPEATED -- SECOND VERSION DELETED.
          (4) TWO USELESS INITIALIZATIONS DELETED.


ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER.
    CYBER-74.
OBJECT-COMPUTER.
    CYBER-74.
SPECIAL-NAMES.
    C01 IS TO-TOP-OF-PAGE.

INPUT-OUTPUT SECTION.
```

```
FILE-CONTROL.
    SELECT CARD-FILE ASSIGN TO INPUTO.
    SELECT PRINT-FILE ASSIGN TO OUTPUTO.

DATA DIVISION.

FILE SECTION.

FD  CARD-FILE
    RECORD CONTAINS 80 CHARACTERS,
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS CARD-RECORD.

01  CARD-RECORD.
    02  I-CARD-DATA.
        03  I-STORE-NUMBER          PIC 99.
        03  I-DEPARTMENT            PIC XX.
        03  I-SALESMAN-NUMBER       PIC 999.
        03  I-SALESMAN-NAME         PIC X(20).
        03  I-YEAR-TO-DATE-SALES    PIC 9(5)V99.
        03  I-CURRENT-SALES         PIC 9(5)V99.
        03  I-COMMISSION-RATE       PIC V99.
        03  I-MONTHS-EMPLOYED       PIC 99.
    02  FILLER                      PIC X(35) .

FD  PRINT-FILE
    RECORD CONTAINS 132 CHARACTERS,
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS LINE-RECORD.

01  LINE-RECORD                     PIC X(132).


WORKING-STORAGE SECTION.

77  W-DEPARTMENT-BONUS              PIC V999.
77  W-VOLUME-BONUS                  PIC V999.
77  W-DEPARTMENT                    PIC XX.
77  W-STORE-NUMBER                  PIC 99.
77  W-SALESMAN-NUMBER               PIC 999.
77  W-YEAR-TO-DATE-SALES            PIC 9(5)V99.
77  W-CURRENT-SALES                 PIC 9(5)V99.
77  W-COMMISSION-RATE               PIC V99.
77  W-MONTHS-EMPLOYED               PIC 99.
77  W-CURRENT-COMMISSION            PIC 9(4)V99.
77  W-TOTAL-YEAR-TO-DATE-SALES      PIC 9(9)V99
    VALUE 0.
77  W-TOTAL-CURRENT-SALES           PIC 9(8)V99
    VALUE 0.
77  W-TOTAL-CURRENT-COMMISSION      PIC 9(7)V99
    VALUE 0.
77  W-AVERAGE-MONTHLY-SALES         PIC 9(7)V99
    VALUE 0.


*01 KEY-TO-RECORDS.
*   02  SALESMAN-NUM                PIC 999.

01  FLAGS.
    02  VALID-DATA-FLAG             PIC  XXX
        VALUE 'YES'.
    02  MORE-DATA-REMAINS-FLAG      PIC  XXX
        VALUE 'YES'.
```

```
01  CONSTANTS.
    02  DEPT.
        03  DEPT-1-OR-2                PIC V999
            VALUE 0.001.
        03  DEPT-6-OR-9                PIC V999
            VALUE 0.004.
        03  DEPT-5-OR-7                PIC V999
            VALUE 0.006.
        03  DEPT-4                     PIC V999
            VALUE 0.007.
        03  DEPT-OTHER                 PIC V999
            VALUE 0.000.
    02 VOLUMN.
        03  LEVEL-1                    PIC V999
            VALUE 0.
        03  LEVEL-2                    PIC V999
            VALUE 0.003.
        03  LEVEL-3                    PIC V999
            VALUE 0.004.
        03  LEVEL-4                    PIC V999
            VALUE 0.006.

01  COUNTERS.
    02  LINE-COUNT                     PIC 99
        VALUE 0.

01  FINAL-TOTAL-LINE.
    02  FILLER                         PIC X(10)
        VALUE '    TOTAL'.
    02  FILLER                         PIC X(51)
        VALUE SPACES.
    02  O-TOTAL-YEAR-TO-DATE-SALES PIC Z(9).99.
    02  FILLER                         PIC XXX
        VALUE SPACES.
    02  O-TOTAL-CURRENT-SALES          PIC Z(8).99.
    02  FILLER                         PIC X(15)
        VALUE SPACES.
    02  O-TOTAL-CURRENT-COMMISSION PIC Z(7).99.
    02  FILLER                         PIC X(20)
        VALUE SPACES.

01  REPORT-LINE-1.
    02  FILLER                         PIC X(61)
        VALUE SPACES.
    02  FILLER                         PIC X(10)
        VALUE 'COMMISSION'.
    02  FILLER                         PIC X(50)
        VALUE SPACES.
    02  FILLER                         PIC X(6)
        VALUE 'PAGE  '.
    02  O-PAGE-NUMBER                  PIC 999
        VALUE 0.
    02  FILLER                         PIC XX
        VALUE SPACES.

01  REPORT-LINE-2.
    02  FILLER                         PIC X(63)
        VALUE SPACES.
    02  FILLER                         PIC X(6)
        VALUE 'REPORT'.
    02  FILLER                         PIC X(63)
        VALUE SPACES.

01  HEADING-LINE-1.
```

```
           02  FILLER                          PIC  X(4)
               VALUE SPACES.
           02  FILLER                          PIC  X(5)
               VALUE 'STORE'.
           02  FILLER                          PIC  X(4)
               VALUE SPACES.
           02  FILLER                          PIC  X(10)
               VALUE 'DEPARTMENT'.
           02  FILLER                          PIC  X(4)
               VALUE SPACES.
           02  FILLER                          PIC  X(8)
               VALUE 'SALESMAN'.
           02  FILLER                          PIC  X(9)
               VALUE SPACES.
           02  FILLER                          PIC  X(8)
               VALUE 'SALESMAN'.
           02  FILLER                          PIC  X(10)
               VALUE SPACES.
           02  FILLER                          PIC  X(12)
               VALUE 'YEAR TO DATE'.
           02  FILLER                          PIC  X(5)
               VALUE SPACES.
           02  FILLER                          PIC  X(7)
               VALUE 'CURRENT'.
           02  FILLER                          PIC  X(4)
               VALUE SPACES.
           02  FILLER                          PIC  X(10)
               VALUE 'COMMISSION'.
           02  FILLER                          PIC  X(5)
               VALUE SPACES.
           02  FILLER                          PIC  X(7)
               VALUE 'CURRENT'.
           02  FILLER                          PIC  X(6)
               VALUE SPACES.
           02  FILLER                          PIC  X(6)
               VALUE 'MONTHS'.
           02  FILLER                          PIC  X(8)
               VALUE SPACES.

       01  HEADING-LINE-2.
           02  FILLER                          PIC  X(4)
               VALUE SPACES.
           02  FILLER                          PIC  X(6)
               VALUE 'NUMBER'.
           02  FILLER                          PIC  X(18)
               VALUE SPACES.
           02  FILLER                          PIC  X(6)
               VALUE 'NUMBER'.
           02  FILLER                          PIC  X(12)
               VALUE SPACES.
           02  FILLER                          PIC  X(4)
               VALUE 'NAME'.
           02  FILLER                          PIC  X(16)
               VALUE SPACES.
           02  FILLER                          PIC  X(5)
               VALUE 'SALES'.
           02  FILLER                          PIC  X(9)
               VALUE SPACES.
           02  FILLER                          PIC  X(5)
               VALUE 'SALES'.
           02  FILLER                          PIC  X(8)
               VALUE SPACES.
           02  FILLER                          PIC  X(4)
               VALUE 'RATE'.
```

```
        02  FILLER                          PIC  X(7)
            VALUE SPACES.
        02  FILLER                          PIC  X(10)
            VALUE 'COMMISSION'.
        02  FILLER                          PIC  X(3)
            VALUE SPACES.
        02  FILLER                          PIC  X(8)
            VALUE 'EMPLOYED'.
        02  FILLER                          PIC  X(7)
            VALUE SPACES.

    01  VALID-DATA-LINE.
        02  FILLER                          PIC X(6)
            VALUE SPACES.
        02  O-STORE-NUMBER                  PIC Z9.
        02  FILLER                          PIC X(9)
            VALUE SPACES.
        02  O-DEPARTMENT                    PIC XX.
        02  FILLER                          PIC X(10)
            VALUE SPACES.
        02  O-SALESMAN-NUMBER               PIC ZZ9.
        02  FILLER                          PIC X(6)
            VALUE SPACES.
        02  O-SALESMAN-NAME                 PIC X(20).
        02  FILLER                          PIC X(6)
            VALUE SPACES.
        02  O-YEAR-TO-DATE-SALES            PIC Z(6).99.
        02  FILLER                          PIC X(5)
            VALUE SPACES.
        02  O-CURRENT-SALES                 PIC Z(6).99.
        02  FILLER                          PIC X(7)
            VALUE SPACES.
        02  O-COMMISSION-RATE               PIC .99.
        02  FILLER                          PIC X(7)
            VALUE SPACES.
        02  O-CURRENT-COMMISSION            PIC Z(5).99.
        02  FILLER                          PIC X(8)
            VALUE SPACES.
        02  O-MONTHS-EMPLOYED               PIC Z9.
        02  FILLER                          PIC X(10)
            VALUE SPACES.

    01  INVALID-DATA-LINE.
        02  O-BAD-DATA                      PIC  X(45).
        02  FILLER                          PIC  X(30)
            VALUE '     INVALID DATA ON THIS CARD'.
        02  FILLER                          PIC  X(57)
            VALUE SPACES.



    PROCEDURE DIVISION.


    PREPARE-PAYMENT-REPORT.
        OPEN INPUT CARD-FILE
            OUTPUT PRINT-FILE.
        READ CARD-FILE
            AT END MOVE 'NO' TO MORE-DATA-REMAINS-FLAG.

        IF MORE-DATA-REMAINS-FLAG = 'YES'
            PERFORM REPORT-HEADER-OUTPUT
            PERFORM HEADING-OUTPUT
```

```
          PERFORM COMMISSION-CALCULATION
              UNTIL MORE-DATA-REMAINS-FLAG = 'NO '.

     PERFORM CALCULATED-TOTALS-OUTPUT.
     CLOSE CARD-FILE
          PRINT-FILE.
     STOP RUN.


*  CHECK VARIABLES TO SEE IF THEY CONTAIN VALID INFORMATION

 VALIDATION.
     IF I-STORE-NUMBER IS NUMERIC
       AND I-SALESMAN-NUMBER IS NUMERIC
       AND I-YEAR-TO-DATE-SALES IS NUMERIC
       AND I-CURRENT-SALES IS NUMERIC
       AND I-COMMISSION-RATE IS NUMERIC
       AND I-MONTHS-EMPLOYED IS NUMERIC
          MOVE 'YES' TO VALID-DATA-FLAG
     ELSE
          MOVE 'NO' TO VALID-DATA-FLAG.


*  MOVE INPUT INFORMATION TO WORKING STORAGE
*  VARIABLES

 DATA-MOVE.
     MOVE I-STORE-NUMBER TO W-STORE-NUMBER.
     MOVE I-DEPARTMENT TO W-DEPARTMENT.
     MOVE I-SALESMAN-NUMBER TO W-SALESMAN-NUMBER.
     MOVE I-YEAR-TO-DATE-SALES TO W-YEAR-TO-DATE-SALES.
     MOVE I-CURRENT-SALES TO W-CURRENT-SALES.
     MOVE I-COMMISSION-RATE TO W-COMMISSION-RATE.
     MOVE I-MONTHS-EMPLOYED TO W-MONTHS-EMPLOYED.

 CALCULATE-DEPARTMENT-BONUS.
     IF W-DEPARTMENT = '01' OR
        W-DEPARTMENT = '02'
          MOVE DEPT-1-OR-2 TO W-DEPARTMENT-BONUS
     ELSE IF W-DEPARTMENT = '06' OR
               W-DEPARTMENT = '09'
          MOVE DEPT-6-OR-9 TO W-DEPARTMENT-BONUS
     ELSE IF W-DEPARTMENT = '05' OR
               W-DEPARTMENT = '07'
          MOVE DEPT-5-OR-7 TO W-DEPARTMENT-BONUS
     ELSE IF W-DEPARTMENT = '04'
          MOVE DEPT-4 TO W-DEPARTMENT-BONUS
     ELSE
          MOVE DEPT-OTHER TO W-DEPARTMENT-BONUS.

 CALCULATE-VOLUME-BONUS.
     COMPUTE W-AVERAGE-MONTHLY-SALES ROUNDED =
       ( W-YEAR-TO-DATE-SALES + W-CURRENT-SALES )
       / W-MONTHS-EMPLOYED.
     IF W-AVERAGE-MONTHLY-SALES < 500
          MOVE LEVEL-1 TO W-VOLUME-BONUS
     ELSE IF W-AVERAGE-MONTHLY-SALES < 999.99
          MOVE LEVEL-2 TO W-VOLUME-BONUS
     ELSE IF W-AVERAGE-MONTHLY-SALES < 1999.99
          MOVE LEVEL-3 TO W-VOLUME-BONUS
     ELSE
          MOVE LEVEL-4 TO W-VOLUME-BONUS.

 COMMISSION-CALCULATION.
```

```
           PERFORM VALIDATION.

           IF VALID-DATA-FLAG = 'YES'
               PERFORM DATA-MOVE
               PERFORM CALCULATE-DEPARTMENT-BONUS
               PERFORM CALCULATE-VOLUME-BONUS
               COMPUTE W-CURRENT-COMMISSION ROUNDED = W-CURRENT-SALES *
                   ( W-COMMISSION-RATE + W-VOLUME-BONUS +
                   W-DEPARTMENT-BONUS )
               ADD W-YEAR-TO-DATE-SALES TO W-TOTAL-YEAR-TO-DATE-SALES
               ADD W-CURRENT-SALES TO W-TOTAL-CURRENT-SALES
               ADD W-CURRENT-COMMISSION TO W-TOTAL-CURRENT-COMMISSION
               PERFORM VALID-DATA-OUTPUT
           ELSE
               PERFORM INVALID-DATA-OUTPUT.

           READ CARD-FILE
               AT END MOVE 'NO' TO MORE-DATA-REMAINS-FLAG.

       VALID-DATA-OUTPUT.
           MOVE W-STORE-NUMBER TO O-STORE-NUMBER.
           MOVE W-DEPARTMENT TO O-DEPARTMENT.
           MOVE W-SALESMAN-NUMBER TO O-SALESMAN-NUMBER.
           MOVE I-SALESMAN-NAME TO O-SALESMAN-NAME.
           MOVE W-YEAR-TO-DATE-SALES TO O-YEAR-TO-DATE-SALES.
           MOVE W-CURRENT-SALES TO O-CURRENT-SALES.
           MOVE W-COMMISSION-RATE TO O-COMMISSION-RATE.
           MOVE W-CURRENT-COMMISSION TO O-CURRENT-COMMISSION.
           MOVE W-MONTHS-EMPLOYED TO O-MONTHS-EMPLOYED.
      *    MOVE I-SALESMAN-NAME TO O-SALESMAN-NAME.
           MOVE VALID-DATA-LINE TO LINE-RECORD.
           WRITE LINE-RECORD AFTER ADVANCING 1 LINES.
           ADD 1 TO LINE-COUNT.
           IF LINE-COUNT IS GREATER THAN 10
      *        MOVE 0 TO LINE-COUNT
               PERFORM REPORT-HEADER-OUTPUT
               PERFORM HEADING-OUTPUT.

       INVALID-DATA-OUTPUT.
           MOVE I-CARD-DATA TO O-BAD-DATA.
           MOVE INVALID-DATA-LINE TO LINE-RECORD.
           WRITE LINE-RECORD AFTER ADVANCING 1 LINES.
           ADD 1 TO LINE-COUNT.
           IF LINE-COUNT IS GREATER THAN 10
      *        MOVE 0 TO LINE-COUNT
               PERFORM REPORT-HEADER-OUTPUT
               PERFORM HEADING-OUTPUT.

       HEADING-OUTPUT.
           MOVE HEADING-LINE-1 TO LINE-RECORD.
           WRITE LINE-RECORD AFTER ADVANCING 1 LINES.
           MOVE HEADING-LINE-2 TO LINE-RECORD.
           WRITE LINE-RECORD AFTER ADVANCING 1 LINES.
           MOVE SPACES TO LINE-RECORD.
           WRITE LINE-RECORD AFTER ADVANCING 2 LINES.
           ADD 4 TO LINE-COUNT.

       CALCULATED-TOTALS-OUTPUT.
           MOVE W-TOTAL-YEAR-TO-DATE-SALES TO O-TOTAL-YEAR-TO-DATE-SALES
           MOVE W-TOTAL-CURRENT-SALES TO O-TOTAL-CURRENT-SALES.
           MOVE W-TOTAL-CURRENT-COMMISSION TO O-TOTAL-CURRENT-COMMISSION
           MOVE FINAL-TOTAL-LINE TO LINE-RECORD.
           WRITE LINE-RECORD AFTER ADVANCING 2 LINES.
```

```
REPORT-HEADER-OUTPUT.
    ADD 1 TO O-PAGE-NUMBER.
    MOVE REPORT-LINE-1 TO LINE-RECORD.
    WRITE LINE-RECORD AFTER ADVANCING TO-TOP-OF-PAGE.
    MOVE REPORT-LINE-2 TO LINE-RECORD.
    WRITE LINE-RECORD AFTER ADVANCING 1 LINES.
    MOVE SPACES TO LINE-RECORD.
    WRITE LINE-RECORD AFTER ADVANCING 3 LINES.
    MOVE 4 TO LINE-COUNT.
```

PROGRAM 6

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINTMFS.
REMARKS.   THIS PROGRAM IS ADAPTED FROM YOURDAN'S "LEARNING
           TO PROGRAM IN STRUCTURED COBOL".
           (1)    THE PROGRAM AS PUBLISHED DID NOT WORK.  THE LAST
           PAIR OF APPLICATION CARDS WAS IGNORED.  IF THERE
           WAS NO LAST PAIR (EMPTY FILE) THE PROGRAM BOMBED.
           THIS ERROR WAS FIXED BY ADDING ANOTHER FILE-CONTROL
           FLAG AND ADDING LOGIC IN "B1-GET-A-PAIR..."
           (2)    THE NOTE ABOUT CHECKING PAIR VALIDITY
           IN PARAGRAPH "A2-UPDATE MASTER" SHOULD BE REPEATED
           IN THE ANALOGOUS PARAGRAPH "A4-ADD-REMAINING-CARDS".
           (3)    IF THE FIRST CARD IS INVALID, ITS LOG ENTRY
           WOULD HAVE BEEN WRITTEN BEFORE THE LOG FILE HEADER.
           (4)    THE PUBLISHED PROGRAM CONTAINED MUCH EXTRANEOUS
           CODE.  THE REASON FOR SOME OF THIS WAS THE FREE USE OF
           THE "COPY" VERB.  THESE PRODUCED MANY UNNECESSARY
           MUTANTS, AND HAVE BEEN COMMENTED OUT WITH "***".
           (5)    THE PROGRAM DID NOT DO ANYTHING SENSIBLE WHEN
           THE END-OF-FILE WAS ENCOUNTERED AFTER THE FIRST OF A
           PAIR OF CARDS.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  PRIME.
OBJECT-COMPUTER.  PRIME.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT APPLICATION-CARDS-FILE     ASSIGN TO INPUT1.
    SELECT UPDATE-LISTING             ASSIGN TO OUTPUT1.
    SELECT CREDIT-MASTER-OLD-FILE     ASSIGN TO INPUT2.
    SELECT CREDIT-MASTER-NEW-FILE     ASSIGN TO OUTPUT2.

DATA DIVISION.
FILE SECTION.

FD  APPLICATION-CARDS-FILE
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS NAME-ADDRESS-AND-PHONE-IN.
01  NAME-ADDRESS-AND-PHONE-IN.
    05  NAME-AND-ADDRESS-IN.
        10  NAME-IN                          PIC X(20).
***     10  ADDRESS-IN.
***         15  STREET-IN                    PIC X(20).
***         15  CITY-IN                      PIC X(13).
***         15  STATE-IN                     PIC XX.
***         15  ZIP-IN                       PIC X(5).
        10  ADDRESS-IN                       PIC X(40).
    05  PHONE-IN                         PIC X(11).
    05  FILLER                           PIC X.
    05  CHANGE-CODE-IN                   PIC XX.
    05  ACCT-NUM-IN1                     PIC 9(6).

FD  UPDATE-LISTING
    RECORD CONTAINS 132 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS PRINT-LINE-OUT.
01  PRINT-LINE-OUT                       PIC X(132).

FD  CREDIT-MASTER-OLD-FILE
```

```
           RECORD CONTAINS 127 CHARACTERS
           LABEL RECORDS ARE STANDARD
           DATA RECORD IS CREDIT-MASTER-RECORD.
       01  CREDIT-MASTER-OLD-RECORD.
           05  ACCT-NUM-MAS-OLD                PIC 9(6).
      *** THE SUBFIELDS ARE NEVER REFERRRED TO IN THE PROGRAM
      ***  USE FILLER INSTEAD
      ***  05  NAME-AND-ADDRESS-MAS-OLD.
      ***      10  NAME-MAS-OLD                PIC X(20).
      ***      10  STREET-MAS-OLD              PIC X(20).
      ***      10  CITY-MAS-OLD                PIC X(13).
      ***      10  STATE-MAS-OLD               PIC XX.
      ***      10  ZIP-MAS-OLD                 PIC 9(5).
      ***  05  PHONE-MAS-OLD.
      ***      10  AREA-CODE-MAS-OLD           PIC 9(3).
      ***      10  NUMBER-MAS-OLD              PIC 9(7).
      ***********************************************************
           05  FILLER                          PIC X(70).
      *** THE SUBFIELDS ARE NEVER REFERRED TO IN THE PROGRAM.
      ***  05  CREDIT-INFO-MAS-OLD.
      ***      10  SEX-MAS-OLD                 PIC X.
      ***      10  MARITAL-STATUS-MAS-OLD      PIC X.
      ***      10  NUMBER-DEPENS-MAS-OLD       PIC 99.
      ***      10  INCOME-HUNDREDS-MAS-OLD     PIC 9(3).
      ***      10  YEARS-EMPLOYED-MAS-OLD      PIC 99.
      ***      10  OWN-OR-RENT-MAS-OLD         PIC X.
      ***      10  MORGAGE-OR-RENTAL-MAS-OLD   PIC 9(3).
      ***      10  OTHER-PAYMENTS-MAS-OLD      PIC 9(3).
           05  CREDIT-INFO-MAS-OLD             PIC X(16).
           05  ACCOUNT-INFO-MAS-OLD.
      ***      10  DISCR-INCOME-MAS-OLD        PIC S9(3).
      ***      10  CREDIT-LIMIT-OLD            PIC 9(4).
               10  FILLER                      PIC S9(3).
               10  FILLER                      PIC 9(4).
               10  CURRENT-BALANCE-OWING-OLD   PIC S9(6)V99.
           05  SPARE-CHARACTERS-OLD            PIC X(20).

       FD  CREDIT-MASTER-NEW-FILE
           RECORD CONTAINS 127 CHARACTERS
           LABEL RECORDS ARE STANDARD
           DATA RECORD IS CREDIT-MASTER-RECORD.
       01  CREDIT-MASTER-NEW-RECORD.
           05  ACCT-NUM-MAS-NEW                PIC 9(6).
      ***  05  NAME-AND-ADDRESS-MAS-NEW.
      ***      10  NAME-MAS-NEW                PIC X(20).
      ***      10  STREET-MAS-NEW              PIC X(20).
      ***      10  CITY-MAS-NEW                PIC X(13).
      ***      10  STATE-MAS-NEW               PIC XX.
      ***      10  ZIP-MAS-NEW                 PIC 9(5).
           05  NAME-AND-ADDRESS-MAS-NEW        PIC X(60).
           05  PHONE-MAS-NEW.
               10  AREA-CODE-MAS-NEW           PIC 9(3).
               10  NUMBR-MAS-NEW               PIC 9(7).
           05  CREDIT-INFO-MAS-NEW.
               10  SEX-MAS-NEW                 PIC X.
               10  MARITAL-STATUS-MAS-NEW      PIC X.
               10  NUMBER-DEPENS-MAS-NEW       PIC 99.
               10  INCOME-HUNDREDS-MAS-NEW     PIC 9(3).
               10  YEARS-EMPLOYED-MAS-NEW      PIC 99.
               10  OWN-OR-RENT-MAS-NEW         PIC X.
               10  MORGAGE-OR-RENTAL-MAS-NEW   PIC 9(3).
               10  OTHER-PAYMENTS-MAS-NEW      PIC 9(3).
           05  ACCOUNT-INFO-MAS-NEW.
               10  DISCR-INCOME-MAS-NEW        PIC S9(3).
```

```
                10  CREDIT-LIMIT-MAS-NEW        PIC 9(4).
                10  CURRENT-BALANCE-OWING-NEW   PIC S9(6)V99.
            05  SPARE-CHARACTERS-NEW            PIC X(20).

       WORKING-STORAGE SECTION.

       01  CREDIT-INFORMATION-IN.
           05  CARD-TYPE-IN                    PIC X.
           05  ACCT-NUM-IN2                    PIC 9(6).
           05  FILLER                          PIC X.
           05  CREDIT-INFO-IN                  PIC X(22).
           05  FILLER                          PIC X(50).

       01  COMMON-WS.
           05  CARDS-LEFT                      PIC X(3).
           05  NEXT-CARD-THERE                 PIC X(3).
           05  OLD-MASTER-RECORDS-LEFT         PIC X(3).
           05  NEW-MASTER-RECORDS-LEFT         PIC X(3).
           05  FIRST-CARD                      PIC X(4).
           05  SECOND-CARD                     PIC X(4).
           05  ACCT-NUM-MATCH                  PIC X(4).
           05  PAIR-VALIDITY                   PIC X(4).

       01  LOG-HEADER-WSA1.
           05  FILLER                          PIC X(47) VALUE SPACES.
           05  FILLER                          PIC X(38)
                   VALUE 'LOG OF ADDITIONS DELETIONS AND CHANGES'.
           05  FILLER                          PIC X(47) VALUE SPACES.

      ***01  HEADER-WSA5.
      ***    05  FILLER                          PIC X(51)  VALUE SPACES
      ***    05  TITLE                           PIC X(30)
      ***            VALUE 'CONTENTS OF CREDIT MASTER FILE'.
      ***    05  FILLER                          PIC X(51)  VALUE SPACES
       01  APPLICATION-DATA-WSB2.
           05  NAME-AND-ADDRESS-WS.
               10  NAME-WS                     PIC X(20).
      ***      10  ADDRESS-WS.
      ***          15  STREET-WS               PIC X(20).
      ***          15  CITY-WS                 PIC X(13).
      ***          15  STATE-WS                PIC XX.
      ***          15  ZIP-WS                  PIC X(5).
               10  ADDRESS-WS                  PIC X(40).
           05  PHONE-WS .
               10  AREA-CODE-WS                PIC 9(3).
               10  NUMBR-WS                    PIC X(8).
           05  FILLER                          PIC X  VALUE SPACE.
           05  CHANGE-CODE-WS                  PIC XX.
           05  ACCT-NUM-WS                     PIC 9(6).
           05  CREDIT-INFO-WS.
               10  SEX-WS                      PIC X.
      **           88  MALE        VALUE 'M'.
      **           88  FEMALE      VALUE 'F'.
               10  FILLER                      PIC X.
               10  MARITAL-STATUS-WS           PIC X.
      **           88  SINGLE      VALUE 'S'.
      **           88  MARRIED     VALUE 'M'.
      **           88  DIVORCED    VALUE 'D'.
      **           88  WIDOWED     VALUE 'W'.
               10  FILLER                      PIC X.
               10  NUMBER-DEPENS-WS            PIC 9.
               10  FILLER                      PIC X.
               10  INCOME-HUNDREDS-WS          PIC 9(3).
               10  FILLER                      PIC X.
```

```
                10  YEARS-EMPLOYED-WS            PIC 99.
                10  FILLER                       PIC X.
                10  OWN-OR-RENT-WS               PIC X.
**                  88  OWNED       VALUE 'O'.
**                  88  RENTED      VALUE 'R'.
                10  FILLER                       PIC X.
                10  MORGAGE-OR-RENTAL-WS         PIC 9(3).
                10  FILLER                       PIC X.
                10  OTHER-PAYMENTS-WS            PIC 9(3).

       01  UPDATE-MESSAGE-AREA-WSB2.
           05  UPDATE-MESSAGE-AREA               PIC X(15).

       01  CREDIT-MASTER-PRINT-LINE.
           05  FILLER                            PIC X(4)    VALUE SPACES.
           05  CREDIT-MASTER-OUT                 PIC X(128).

       01  UPDATE-RECORD-PRINT-LINE.
           05  FILLER                            PIC X(4)  VALUE SPACES.
           05  APPLICATION-DATA-OUT              PIC X(102).
           05  FILLER                            PIC X(4)  VALUE SPACES.
           05  MESSAGE-AREA-OUT                  PIC X(15).

       01  DISCR-INCOME-CALC-FIELDS-WSC8.
           05  ANNUAL-INCOME-WS                  PIC 9(5).
           05  ANNUAL-TAX-WS                     PIC 9(5).
           05  TAX-RATE-WS                       PIC 9V99    VALUE 0.25.
           05  MONTHS-IN-YEAR                    PIC 99      VALUE 12.
           05  MONTHLY-NET-INCOME-WS             PIC 9(4).
           05  MONTHLY-PAYMENTS-WS               PIC 9(4).
           05  DISCR-INCOME-WS                   PIC S9(3).

       01  CREDIT-LIMIT-CALC-FIELDS-WSC9.
           05  CREDIT-FACTOR                     PIC 9.
           05  FACTOR1                           PIC 9     VALUE 1.
           05  FACTOR2                           PIC 9     VALUE 2.
           05  FACTOR3                           PIC 9     VALUE 3.
           05  FACTOR4                           PIC 9     VALUE 4.
           05  FACTOR5                           PIC 9     VALUE 5.
           05  CREDIT-LIMIT-WS                   PIC 9(4).
           05  UPPER-LIMIT-WS                    PIC 9(4)  VALUE 2500.
*** NEVER USED
***        05  TOTAL-CREDIT-GIVEN-WS             PIC 9(7).

       01  ASSEMBLE-TEL-NUM-WSD1.
           05  TEL-NUMBR-WITH-HYPHEN.
               10  EXCHANGE-IN                   PIC 9(3).
               10  FILLER                        PIC X.
               10  FOUR-DIGIT-NUMBR-IN           PIC 9(4).
           05  TEL-NUMBR-WITHOUT-HYPHEN.
               10  EXCHANGE                      PIC 9(3).
               10  FOUR-DIGIT-NUMBR              PIC 9(4).

       01  CARD-ERROR-LINE1-WS.
           05  FILLER                            PIC X(5)  VALUE SPACES.
           05  FILLER                            PIC X(12)
                   VALUE 'FIRST CARD  '.
           05  FIRST-CARD-ERR1                   PIC X(4).
           05  FILLER                            PIC XX  VALUE SPACES.
           05  NAME-ERR1                         PIC X(20).
           05  ADDRESS-ERR1                      PIC X(40).
           05  PHONE-ERR1                        PIC X(11).
           05  FILLER                            PIC X(3)    VALUE SPACES.
           05  ACCT-NUM-ERR1                     PIC 9(6).
```

```

       01  CARD-ERROR-LINE2-WS.
           05  FILLER                          PIC X(5)   VALUE SPACES.
           05  FILLER                          PIC X(12)
                   VALUE  'SECOND CARD '.
           05  SECOND-CARD-ERR2                PIC X(4).
           05  FILLER                          PIC X(2)    VALUE SPACES.
           05  CREDIT-INFO-ERR2                PIC X(80).
           05  MESSAGE-ERR-LINE-2              PIC X(29)   VALUE SPACES.

       PROCEDURE DIVISION.

       A0-MAIN-BODY.
           PERFORM A1-INITIALIZE.
           PERFORM A2-UPDATE-MASTER
             UNTIL OLD-MASTER-RECORDS-LEFT = 'NO '
               OR CARDS-LEFT = 'NO '.
           IF CARDS-LEFT = 'NO '
      *                                    THERE ARE MORE OLD MASTER REC
               PERFORM A3-COPY-REMAINING-OLD-MASTER
                 UNTIL OLD-MASTER-RECORDS-LEFT = 'NO '
           ELSE
      *                                    THERE ARE NO MORE CARDS, SO
               PERFORM A4-ADD-REMAINING-CARDS
                 UNTIL CARDS-LEFT = 'NO '.
      ****************************************************************
      *    CODE TO LIST THE CONTENTS OF THE NEW MASTER HAS BEEN OMITTED.
      *    IT WOULD HAVE REQUIRED CLOSING THE NEW MASTER AND REOPENING
      *    IT FOR INPUT.  THIS IS BEYOND THE ABILITIES OF CMS.1
      *    THE DELETION AMOUNTS TO ABOUT 20 LINES OF CODE.
      ****************************************************************
           PERFORM A7-END-OF-JOB.
           STOP RUN.

       A1-INITIALIZE.
           OPEN INPUT     APPLICATION-CARDS-FILE
                          CREDIT-MASTER-OLD-FILE
                OUTPUT    CREDIT-MASTER-NEW-FILE
                          UPDATE-LISTING.
      *** USELESS INITIALIZATIONS HAVE BEEN COMMENTED OUT
      ***  MOVE SPACES TO FIRST-CARD.
      ***  MOVE SPACES TO SECOND-CARD.
      ***  MOVE SPACES TO ACCT-NUM-MATCH.
      ***  MOVE SPACES TO PAIR-VALIDITY.
      ***  MOVE ZEROES TO ANNUAL-INCOME-WS.
      ***  MOVE ZEROES TO ANNUAL-TAX-WS.
      ***  MOVE ZEROES TO MONTHLY-NET-INCOME-WS.
      ***  MOVE ZEROES TO MONTHLY-PAYMENTS-WS.
      ***  MOVE ZEROES TO DISCR-INCOME-WS.
      ***  MOVE ZEROES TO CREDIT-FACTOR.
      ***  MOVE ZEROES TO CREDIT-LIMIT-WS.
      ***  MOVE ZEROES TO TOTAL-CREDIT-GIVEN-WS.
           MOVE 'YES' TO CARDS-LEFT.
           MOVE 'YES' TO NEXT-CARD-THERE.
           MOVE 'YES' TO OLD-MASTER-RECORDS-LEFT.
      **  THE FOLLOWING STATEMENT WAS MOVED HERE FROM THE END OF THE
      **  PARAGRAPH, SO THAT THE HEADER WOULD BE WRITTEN BEFORE THE
      **  FIRST LOG RECORD, IF THE FIRST CARD PAIR IS INVALID.
           WRITE PRINT-LINE-OUT FROM LOG-HEADER-WSA1
                              AFTER ADVANCING 3 LINES.
           READ APPLICATION-CARDS-FILE
               AT END MOVE 'NO ' TO NEXT-CARD-THERE.
           PERFORM B1-GET-A-PAIR-OF-CARDS-INTO-WS THRU B1-EXIT.
      * FIRST PAIR OF CARDS IN WS: FIRST CARD OF SECOND PAIR IN BUFFER
```

```
        READ CREDIT-MASTER-OLD-FILE
            AT END MOVE 'NO ' TO OLD-MASTER-RECORDS-LEFT.
* FIRST OLD MASTER RECORD IS IN BUFFER

 A2-UPDATE-MASTER.
* BEFORE COMPARING THE UPDATE WITH THE MASTER, WE MUST CHECK
* THAT WE HAVE A VALID PAIR OF CARDS - IF YOUR PROGRAM DOES
* NOT MAKE THIS TEST, IT WILL ONLY WORK WITH VALID PAIRS OF
* CARDS.
        IF  PAIR-VALIDITY = 'BAD '
            PERFORM B1-GET-A-PAIR-OF-CARDS-INTO-WS THRU B1-EXIT
        ELSE IF ACCT-NUM-WS IS GREATER THAN ACCT-NUM-MAS-OLD
*                    ACCT-NUM-WS IS CARD ACCOUNT NUMBER
            MOVE CREDIT-MASTER-OLD-RECORD TO
                CREDIT-MASTER-NEW-RECORD
            WRITE CREDIT-MASTER-NEW-RECORD
            READ CREDIT-MASTER-OLD-FILE
                AT END MOVE 'NO ' TO OLD-MASTER-RECORDS-LEFT
        ELSE IF ACCT-NUM-WS = ACCT-NUM-MAS-OLD
            PERFORM B2-CHANGE-OR-DELETE-MASTER
            PERFORM B1-GET-A-PAIR-OF-CARDS-INTO-WS THRU B1-EXIT
            READ CREDIT-MASTER-OLD-FILE
                AT END MOVE 'NO ' TO OLD-MASTER-RECORDS-LEFT
        ELSE
*                                    ACCT-NUM-WS IS LESS THAN
*                                    ACCT-NUM-MAS-OLD
            PERFORM B3-ADD-NEW-MASTER
            PERFORM B1-GET-A-PAIR-OF-CARDS-INTO-WS THRU B1-EXIT.

 A3-COPY-REMAINING-OLD-MASTER.
        MOVE CREDIT-MASTER-OLD-RECORD TO
            CREDIT-MASTER-NEW-RECORD
        WRITE CREDIT-MASTER-NEW-RECORD.
        READ CREDIT-MASTER-OLD-FILE
            AT END MOVE 'NO ' TO OLD-MASTER-RECORDS-LEFT.

 A4-ADD-REMAINING-CARDS.
        IF PAIR-VALIDITY = 'BAD ' NEXT SENTENCE
        ELSE PERFORM B3-ADD-NEW-MASTER.
        PERFORM B1-GET-A-PAIR-OF-CARDS-INTO-WS THRU B1-EXIT.

 A7-END-OF-JOB.
        CLOSE APPLICATION-CARDS-FILE
              CREDIT-MASTER-OLD-FILE
              CREDIT-MASTER-NEW-FILE
              UPDATE-LISTING.

 B1-GET-A-PAIR-OF-CARDS-INTO-WS.
        IF NEXT-CARD-THERE = 'NO '
           MOVE 'NO ' TO CARDS-LEFT
           GO TO B1-EXIT.
        PERFORM C1-EDIT-FIRST-CARD.
        PERFORM C2-MOVE-FIRST-CARD-TO-WS.
        READ APPLICATION-CARDS-FILE INTO CREDIT-INFORMATION-IN
            AT END MOVE 'NO ' TO CARDS-LEFT,
                   MOVE SPACES TO  CREDIT-INFORMATION-IN
                                   ACCT-NUM-MATCH
                   MOVE 'NONE' TO SECOND-CARD
                   PERFORM C4-FLUSH-CARDS-TO-ERROR-LINES
                   GO TO B1-EXIT.
        PERFORM C3-EDIT-SECOND-CARD.
        IF (FIRST-CARD = 'GOOD')
            AND (SECOND-CARD = 'GOOD')
            AND (ACCT-NUM-MATCH = 'GOOD')
```

```
                MOVE 'GOOD' TO PAIR-VALIDITY
                MOVE CREDIT-INFO-IN TO CREDIT-INFO-WS
        ELSE
            MOVE 'BAD ' TO PAIR-VALIDITY
            PERFORM C4-FLUSH-CARDS-TO-ERROR-LINES.
        READ APPLICATION-CARDS-FILE
            AT END MOVE 'NO ' TO NEXT-CARD-THERE.

    B1-EXIT.  EXIT.

    B2-CHANGE-OR-DELETE-MASTER.
        IF  CHANGE-CODE-WS = 'CH'
            PERFORM C5-MERGE-UPDATE-WITH-OLD-MAST
            MOVE 'RECORD CHANGED' TO UPDATE-MESSAGE-AREA
            PERFORM C6-LOG-ACTION
            WRITE CREDIT-MASTER-NEW-RECORD
        ELSE IF CHANGE-CODE-WS = 'DE'
*                                   CHECK IF DELETE IS VALID
                IF  CREDIT-INFO-WS IS EQUAL TO SPACES
                    MOVE 'RECORD DELETED' TO UPDATE-MESSAGE-AREA
                    PERFORM C6-LOG-ACTION
                ELSE
                    MOVE 'REC NOT DELETED' TO UPDATE-MESSAGE-AREA
                    MOVE CREDIT-MASTER-OLD-RECORD TO
                    CREDIT-MASTER-NEW-RECORD
                    PERFORM C6-LOG-ACTION
                    WRITE CREDIT-MASTER-NEW-RECORD
        ELSE
            MOVE 'BAD CHANGE CODE' TO UPDATE-MESSAGE-AREA
            MOVE CREDIT-MASTER-OLD-RECORD TO CREDIT-MASTER-NEW-RECORD
            PERFORM C6-LOG-ACTION
            WRITE CREDIT-MASTER-NEW-RECORD.

    B3-ADD-NEW-MASTER.
        PERFORM C8-CALC-DISCRETNRY-INCOME.
        PERFORM C9-CALC-CREDIT-LIMIT.
        PERFORM C10-ASSEMBLE-NEW-MASTER-RECORD.
        MOVE 'RECORD ADDED ' TO UPDATE-MESSAGE-AREA.
        PERFORM C6-LOG-ACTION.
        WRITE CREDIT-MASTER-NEW-RECORD.

    C1-EDIT-FIRST-CARD.
        MOVE 'GOOD' TO FIRST-CARD.
        IF NAME-IN IS EQUAL TO SPACES
            MOVE '*** NAME MISSING ***' TO NAME-IN
            MOVE 'BAD ' TO FIRST-CARD.
        IF ADDRESS-IN IS EQUAL TO SPACES
            MOVE '*** ADDRESS MISSING ***' TO ADDRESS-IN
            MOVE 'BAD ' TO FIRST-CARD.
        IF PHONE-IN IS EQUAL TO SPACES
            MOVE 'NO PHONE **' TO PHONE-IN
            MOVE 'BAD ' TO FIRST-CARD.

    C2-MOVE-FIRST-CARD-TO-WS.
        MOVE NAME-IN TO NAME-WS.
        MOVE ADDRESS-IN TO ADDRESS-WS.
        MOVE PHONE-IN TO PHONE-WS.
        MOVE CHANGE-CODE-IN TO CHANGE-CODE-WS.
        MOVE ACCT-NUM-IN) TO ACCT-NUM-WS.

    C3-EDIT-SECOND-CARD.
        MOVE 'GOOD' TO SECOND-CARD.
        MOVE 'GOOD' TO ACCT-NUM-MATCH.
        IF  CARD-TYPE-IN IS NOT EQUAL TO 'C'
```

```
             MOVE 'BAD ' TO SECOND-CARD.
         IF  ACCT-NUM-IN2 IS NOT EQUAL TO ACCT-NUM-WS
             MOVE 'BAD ' TO ACCT-NUM-MATCH.

 C4-FLUSH-CARDS-TO-ERROR-LINES.
         MOVE FIRST-CARD TO FIRST-CARD-ERR1.
         MOVE NAME-WS TO NAME-ERR1.
         MOVE ADDRESS-WS TO ADDRESS-ERR1.
         MOVE PHONE-WS TO PHONE-ERR1.
         MOVE ACCT-NUM-WS TO ACCT-NUM-ERR1.
         MOVE SECOND-CARD TO SECOND-CARD-ERR2.
**        MOVE CREDIT-INFO-WS TO CREDIT-INFO-ERR2.
**  THE PREVIOUS LINE WAS IN ERROR (BY A SINGLE MUTATION) IN THE
**  PUBLISHED PROGRAM.  THE CORRECT STATEMENT IS:
         MOVE CREDIT-INFO-IN TO CREDIT-INFO-ERR2.
         IF  ACCT-NUM-MATCH = 'BAD '
             MOVE 'ACCOUNT NUMBERS DO NOT MATCH'
                             TO MESSAGE-ERR-LINE-2
         ELSE
             MOVE SPACES TO MESSAGE-ERR-LINE-2.
***      MOVE SPACES TO PRINT-LINE-OUT.
         WRITE PRINT-LINE-OUT FROM CARD-ERROR-LINE1-WS
                 AFTER ADVANCING 3 LINES.
***      MOVE SPACES TO PRINT-LINE-OUT.
         WRITE PRINT-LINE-OUT FROM CARD-ERROR-LINE2-WS
                 AFTER ADVANCING 1 LINES.


 C5-MERGE-UPDATE-WITH-OLD-MAST.
         MOVE ACCT-NUM-MAS-OLD TO ACCT-NUM-MAS-NEW.
         MOVE NAME-AND-ADDRESS-WS TO NAME-AND-ADDRESS-MAS-NEW.
         MOVE AREA-CODE-WS TO AREA-CODE-MAS-NEW.
         PERFORM D1-REMOVE-HYPHEN-FROM-TEL-NUM.
* THE SECOND INPUT CARD HAS CREDIT DATA, IF THIS HAS TO BE
* UPDATED THEN THE DISCRETIONARY INCOME CALC HAS TO BE RUN
         IF CREDIT-INFO-WS IS EQUAL TO SPACES
             MOVE CREDIT-INFO-MAS-OLD TO CREDIT-INFO-MAS-NEW
             MOVE ACCOUNT-INFO-MAS-OLD TO ACCOUNT-INFO-MAS-NEW
         ELSE
             PERFORM C8-CALC-DISCRETNRY-INCOME
             PERFORM C9-CALC-CREDIT-LIMIT
             MOVE SEX-WS                 TO SEX-MAS-NEW
             MOVE MARITAL-STATUS-WS      TO MARITAL-STATUS-MAS-NEW
             MOVE NUMBER-DEPENS-WS       TO NUMBER-DEPENS-MAS-NEW
             MOVE INCOME-HUNDREDS-WS     TO INCOME-HUNDREDS-MAS-NEW
             MOVE YEARS-EMPLOYED-WS      TO YEARS-EMPLOYED-MAS-NEW
             MOVE OWN-OR-RENT-WS         TO OWN-OR-RENT-MAS-NEW
             MOVE MORGAGE-OR-RENTAL-WS   TO MORGAGE-OR-RENTAL-MAS-NEW
             MOVE OTHER-PAYMENTS-WS      TO OTHER-PAYMENTS-MAS-NEW
             MOVE DISCR-INCOME-WS        TO DISCR-INCOME-MAS-NEW
             MOVE CREDIT-LIMIT-WS        TO CREDIT-LIMIT-MAS-NEW.
         MOVE CURRENT-BALANCE-OWING-OLD TO CURRENT-BALANCE-OWING-NEW.
         MOVE SPARE-CHARACTERS-OLD TO SPARE-CHARACTERS-NEW.

 C6-LOG-ACTION.
         IF CHANGE-CODE-WS = 'CH'
*                                   WRITE OLD TAPE RECORD
*                                   WRITE CARD CONTENTS & MESSAGE
*                                   WRITE NEW TAPE RECORD
***          MOVE SPACES TO CREDIT-MASTER-PRINT-LINE
             MOVE CREDIT-MASTER-OLD-RECORD TO CREDIT-MASTER-OUT
             WRITE PRINT-LINE-OUT FROM CREDIT-MASTER-PRINT-LINE
                         AFTER ADVANCING 3 LINES
***          MOVE SPACES TO UPDATE-RECORD-PRINT-LINE
```

```
               MOVE APPLICATION-DATA-WSB2 TO APPLICATION-DATA-OUT
               MOVE UPDATE-MESSAGE-AREA TO MESSAGE-AREA-OUT
               WRITE PRINT-LINE-OUT FROM UPDATE-RECORD-PRINT-LINE
                           AFTER ADVANCING 1 LINES
      ***      MOVE SPACES TO CREDIT-MASTER-PRINT-LINE
               MOVE CREDIT-MASTER-NEW-RECORD TO CREDIT-MASTER-OUT
               WRITE PRINT-LINE-OUT FROM CREDIT-MASTER-PRINT-LINE
                           AFTER ADVANCING 1 LINES
           ELSE  IF CHANGE-CODE-WS = 'DE'
      *                                    WRITE OLD TAPE RECORD
      *                                    WRITE CARD CONTENTS & MESSAGE
      ***      MOVE SPACES TO CREDIT-MASTER-PRINT-LINE
               MOVE CREDIT-MASTER-OLD-RECORD TO CREDIT-MASTER-OUT
               WRITE PRINT-LINE-OUT FROM CREDIT-MASTER-PRINT-LINE
                           AFTER ADVANCING 3 LINES
      ***      MOVE SPACES TO UPDATE-RECORD-PRINT-LINE
               MOVE APPLICATION-DATA-WSB2 TO APPLICATION-DATA-OUT
               MOVE UPDATE-MESSAGE-AREA TO MESSAGE-AREA-OUT
               WRITE PRINT-LINE-OUT FROM UPDATE-RECORD-PRINT-LINE
                           AFTER ADVANCING 1 LINES
           ELSE IF CHANGE-CODE-WS = '  '
      *                                    WRITE CARDS FOR ADDITION
      *                                    WRITE NEW TAPE RECORD
      ***      MOVE SPACES TO UPDATE-RECORD-PRINT-LINE
               MOVE APPLICATION-DATA-WSB2 TO APPLICATION-DATA-OUT
               MOVE UPDATE-MESSAGE-AREA TO MESSAGE-AREA-OUT
               WRITE PRINT-LINE-OUT FROM UPDATE-RECORD-PRINT-LINE
                           AFTER ADVANCING 3 LINES
      ***      MOVE SPACES TO CREDIT-MASTER-PRINT-LINE
               MOVE CREDIT-MASTER-NEW-RECORD TO CREDIT-MASTER-OUT
               WRITE PRINT-LINE-OUT FROM CREDIT-MASTER-PRINT-LINE
                           AFTER ADVANCING 1 LINES

           ELSE
      *                                    WRITE CARD CONTENTS & MESSAGE
               MOVE APPLICATION-DATA-WSB2 TO APPLICATION-DATA-OUT
               MOVE UPDATE-MESSAGE-AREA TO MESSAGE-AREA-OUT
               WRITE PRINT-LINE-OUT FROM UPDATE-RECORD-PRINT-LINE
                           AFTER ADVANCING 3 LINES.

       C8-CALC-DISCRETNRY-INCOME.
           COMPUTE ANNUAL-INCOME-WS = INCOME-HUNDREDS-WS * 100.
           COMPUTE ANNUAL-TAX-WS     = ANNUAL-INCOME-WS * TAX-RATE-WS.
           COMPUTE MONTHLY-NET-INCOME-WS ROUNDED
               = (ANNUAL-INCOME-WS - ANNUAL-TAX-WS) / MONTHS-IN-YEAR.
           COMPUTE MONTHLY-PAYMENTS-WS = MORGAGE-OR-RENTAL-WS
                                + OTHER-PAYMENTS-WS.
           COMPUTE DISCR-INCOME-WS = MONTHLY-NET-INCOME-WS
                                - MONTHLY-PAYMENTS-WS
               ON SIZE ERROR MOVE 999 TO DISCR-INCOME-WS.
      *  DISCRETIONARY INCOMES OVER $999 PER MONTH ARE SET AT $999.

       C9-CALC-CREDIT-LIMIT.
      *    MARRIED?          Y Y Y Y N N N N   THIS DECISION TABLE      *
      *    OWNED?            Y Y N N Y Y N N   SETS OUT COMPANY POLICY  *
      *    2 OR MORE YEARS?  Y N Y N Y N Y N   FOR DETERMINING CREDIT   *
      *    -------------------------------    LIMIT FROM DISCRETIONARY*
      *    CREDIT    FACTOR1             X X   INCOME. FACTOR1 ETC ARE  *
      *    LIMIT           2       X   X       SET UP IN WSC9.          *
      *    MULTIPLE        3         X                                  *
      *    OF DISCR.       4   X X                                      *
      *    INCOME          5 X                                          *
           IF MARITAL-STATUS-WS = 'M'
               IF OWN-OR-RENT-WS = 'O'
```

```
                IF  YEARS-EMPLOYED-WS IS NOT LESS THAN 02
                    MOVE FACTOR5 TO CREDIT-FACTOR
                ELSE
                    MOVE FACTOR4 TO CREDIT-FACTOR
            ELSE
                IF  YEARS-EMPLOYED-WS IS NOT LESS THAN 02
                    MOVE FACTOR4 TO CREDIT-FACTOR
                ELSE
                    MOVE FACTOR2 TO CREDIT-FACTOR
        ELSE
            IF OWN-OR-RENT-WS = 'O'
                IF  YEARS-EMPLOYED-WS IS NOT LESS THAN 02
                    MOVE FACTOR3 TO CREDIT-FACTOR
                ELSE
                    MOVE FACTOR2 TO CREDIT-FACTOR
            ELSE
                MOVE FACTOR1 TO CREDIT-FACTOR.
        COMPUTE CREDIT-LIMIT-WS = DISCR-INCOME-WS * CREDIT-FACTOR.
        IF CREDIT-LIMIT-WS IS GREATER THAN UPPER-LIMIT-WS
            MOVE UPPER-LIMIT-WS TO CREDIT-LIMIT-WS.
***     ADD CREDIT-LIMIT-WS TO TOTAL-CREDIT-GIVEN-WS.

    C10-ASSEMBLE-NEW-MASTER-RECORD.
        MOVE ACCT-NUM-WS TO ACCT-NUM-MAS-NEW.
        MOVE NAME-AND-ADDRESS-WS TO NAME-AND-ADDRESS-MAS-NEW.
        MOVE AREA-CODE-WS TO AREA-CODE-MAS-NEW.
        PERFORM D1-REMOVE-HYPHEN-FROM-TEL-NUM.
        MOVE SEX-WS                  TO SEX-MAS-NEW
        MOVE MARITAL-STATUS-WS       TO MARITAL-STATUS-MAS-NEW
        MOVE NUMBER-DEPENS-WS        TO NUMBER-DEPENS-MAS-NEW
        MOVE INCOME-HUNDREDS-WS      TO INCOME-HUNDREDS-MAS-NEW
        MOVE YEARS-EMPLOYED-WS       TO YEARS-EMPLOYED-MAS-NEW
        MOVE OWN-OR-RENT-WS          TO OWN-OR-RENT-MAS-NEW
        MOVE MORGAGE-OR-RENTAL-WS    TO MORGAGE-OR-RENTAL-MAS-NEW
        MOVE OTHER-PAYMENTS-WS       TO OTHER-PAYMENTS-MAS-NEW.
        MOVE DISCR-INCOME-WS TO DISCR-INCOME-MAS-NEW.
        MOVE CREDIT-LIMIT-WS TO CREDIT-LIMIT-MAS-NEW.
        MOVE ZEROES TO CURRENT-BALANCE-OWING-NEW.
        MOVE SPACES TO SPARE-CHARACTERS-NEW.

    D1-REMOVE-HYPHEN-FROM-TEL-NUM.
        MOVE NUMBR-WS TO TEL-NUMBR-WITH-HYPHEN
        MOVE EXCHANGE-IN TO EXCHANGE
        MOVE FOUR-DIGIT-NUMBR-IN TO FOUR-DIGIT-NUMBR
        MOVE TEL-NUMBR-WITHOUT-HYPHEN TO NUMBR-MAS-NEW.
```

BIBLIOGRAPHY

[1] A. Acree, T. Budd, R. DeMillo, R. Lipton, and F. Sayward, "Mutation Analysis", Georgia Institute of Technology Technical Report GIT-ICS-79/08, September, 1979.

[2] Fortran Automated Verification System (FAVS), Volume I, User's Manual, General Research Corp., Santa Barbara, Ca., Jan. 1979.

[3] D. Baldwin and F. Sayward, "Heuristics for Determining Equivalence of Program Mutations," Yale University, Department of Computer Science Research Report, No. 276, 1979.

[4] R.S. Boyer, B. Elspas, and K.N. Levitt, "SELECT - A Formal System for Testing and Debugging Programs by Symbolic Execution", in Proc. Int. Conf. on Reliable Software, Apr. 1975, pp 234-244.

[5] T. Budd, R.A.DeMillo, R.J. Lipton, and F.G. Sayward, "The Design of a Prototype Mutation System for Program Testing," Proc. 1978 NCC, AFIPS Conference Record, pp. 623-627.

[6] T.A. Budd, R.A. DeMillo, R.J. Lipton, and F.G. Sayward, "Theoretical and Empirical Studies in Program Mutation to Test the Functional Correctness of Programs", submitted for publication, 1979.

[7] J. Burns, "The stability of Test Data from Program Mutation," Digest for the Workshop on Software Testing and Test Documentation , Fort Lauderdale, Fla, 1978, pp. 324-334.

[8] L.A. Clark, "A system to Generate Test Data and Symbolically Execute Programs", IEEE Transactions on Software Engineering, Vol 2, Sept '76,pp 215-222.

[9] L.A. Clark, "Automatic Test DAta Selection Techniques", Software Testing, Volume 2, Infotech International, 1979, pp 43-63.

[10] R.A. DeMillo, R.J. Lipton and A.J. Perlis, "Social Processes and Proofs of Theorems and Programs," CACM, Vol 22(5), (May, 1979), pp. 271-280.

[11] R.A. DeMillo, R.J. Lipton and F.G. Sayward, "Hints on Test Data Selection:Help for the Practicing Programmer," Computer, April, 1978, pp. 34-41.

[12] R.A. DeMillo, R.J. Lipton and F.G. Sayward, "Program Mutation: A New Approach to Program Testing," INFOTECH State of the Art Report on Software Testing, Vol. 2, INFOTECH/SRA, 1979, pp. 107-127 [Note: also see commentaries in Volume 1].

[13] T. Gilb, Software Metrics, Winthrop, 1977.

[14] J. Goodenough and S. Gerhart, "Toward a Theory of Test Data Selection," IEEE Trans. Software Engin., Vol SE-1 , (June, 1975), pp. 156-173.

[15] Concepts of Automated Testing Analysis, (RP-1), Software Technology Center, Science Applications, Inc., San Francisco, Ca.

[16] W.E. Howden, "Reliability of the Path Analysis Testing Strategy," IEEE Trans. Software Engineering, Vol. SE-2(3) (September, 1976), pp. 208-214.

[17] W.E. Howden, "An Evaluation of the Effectiveness of Symbolic Testing," Software Practice and Experience, Volume 8, (1978), pp. 381-397.

[18] "A New Approach to Program Testing", in Proc. Int. Conf. Reliable Software, Apr. 1975, pp 228-233.

[19] R.J. Lipton and F.G. Sayward, "The Status of Research on Program Mutation," Digest of the Workshop on Software Testing and Test Documentation," Fort Lauderdale, Fla, 1978, pp. 355-373.

[20] Z. Manna and R. Waldinger, "The Logic of Computer Programming", IEEE Transactions on Software Engineering, Vol SE-4(3), (September, 1978), pp199-229.

[21] W.D. Maurer, letter in "ACM Forum", Communications of the ACM, vol. 22 no. 11, Nov 1979, pp 525-529.

[22] E.F. Miller, Jr., Methodology for Comprehensive Software Testing, General Research Corporation, Santa Barbara, CA, June 1975

[23] D.C. Montgomery, Design and Analysis of Experiments, Wiley, New York, 1976

[24] L.J. Osterweil and L.D. Fosdick, "Experience with DAVE-- A Fortran Program Analyzer, Proc. 1976 NCC, AFIPS Conference Record, pp 909-915.

[25] L.J. Osterweil and L.D. Fosdick, "Data Flow Analysis as an Aid in Documentation, Assertion Generation, Validation and Error Detection, University of Colorado, Department of Computer Science, Technical Report No. CU-CS-055-74, 1974.

[26 R.A. Overbeek and W.E. Singletqary, ANS Cobol: A Pragmatic Approach, McGraw-Hill, New York, 1975.

[27] M.R. Paige, "Program Graphs, an Algebra, and Their Implication for Programming", IEEE Transactions on Software Engineering, Sept.75, pp286-291.

[28] PRIME Fortran Programmer's Guide, PDR3057, PRIME Computer, Inc. Framingham, Mass. p 4-5.

[29] J.H. Rowland and P.J. Davis, "On the use of Trancendentals for Program Testing", March 1979, submitted to JACM.

[30] Automated Testing Analyzer for Cobol, Software Technology Center, Science Applications, Inc. San Francisco, Ca., April, 1976.

[31] T.A. Thayer, M. Lipow, E.C. Nelson, Software Reliability, North-Holland, 1978.

[32] E.A. Youngs, "Human Errors in Programming," International Journal of Man-Machine Studies, Volume 6 (1974), pp. 361-376.

[33] E. Yourdan, C. Gane, and T. Sarsan, Learning to Program in Structured Cobol, Yourdan, Inc., New York, 1976

[34] G. Williams, "Program Checking", Proceedings of the SIGPLAN Symposium on Compiler Construction, Denver, Colorado, in SIGPLAN Notices, Vol.14(8), Aug 1979, pp 13-25.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

Field	Entry
1. REPORT NUMBER	GIT-ICS-80/12
2. GOVT ACCESSION NO.	AD-A091029
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle)	On Mutation
5. TYPE OF REPORT & PERIOD COVERED	Interim Technical Report
6. PERFORMING ORG. REPORT NUMBER	GIT-ICS-80/12
7. AUTHOR(s)	Allen Troy Acree, Jr.
8. CONTRACT OR GRANT NUMBER(s)	ARO Grant #DAAG29-80-C-0120 ONR Grant #N00014-79-C-0231
9. PERFORMING ORGANIZATION NAME AND ADDRESS	School of Information and Computer Science Georgia Institute of Technology Atlanta, Georgia 30332
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
11. CONTROLLING OFFICE NAME AND ADDRESS	[illegible] Resea[illegible]709
12. REPORT DATE	AUGUST 1980
13. NUMBER OF PAGES	177+vi
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	
15. SECURITY CLASS. (of this report)	Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	

16. DISTRIBUTION STATEMENT (of this Report)

[illegible] unlimited.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS [illegible]

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

competent programmer assumption, coupling hypothesis, mutant equivalence, mutation, testing, validation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Program Mutation is a method for testing computer programs which is effective at uncovering errors and is less expensive to apply than other techniques. Working mutation systems have demonstrated that mutation analysis can be performed at an attractive cost on realistic programs. In this work, the effectiveness of the method is studied by experiments with programs in the target application spaces.

DATE
FILMED

2-8